



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

**GENERATING ENHANCED NATURAL ENVIRONMENTS
AND TERRAIN FOR INTERACTIVE COMBAT
SIMULATIONS (GENETICS)**

by

William David Wells

September 2005

Dissertation Supervisor:

Rudolph Darken

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY		2. REPORT DATE September 2005	3. REPORT TYPE AND DATES COVERED Doctoral Dissertation	
4. TITLE AND SUBTITLE: Generating Enhanced Natural Environments and Terrain for Interactive Combat Simulations (GENETICS)			5. FUNDING NUMBERS	
6. AUTHOR(S) William David Wells				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT <p>Virtual battlefields devoid of vegetation deprive soldiers of valuable training in the critical aspects of terrain tactics and terrain-based situational awareness, but creating realistic landscapes by hand is notoriously expensive; requiring both proprietary tools and trained artists, hampering rapid scenario generation and limiting reuse.</p> <p>GENETICS is a new object placement scheme where the arduous task of placing vegetation objects is reduced to finding readily-available source data and setting a few parameters. Our approach constructs large-scale natural environments at run-time using a procedural image-based algorithm without the need for skilled artists or proprietary tools. The resulting vegetation-laden terrain looks realistic, and the algorithm can be extended to incorporate a wide variety of environmental factors. GENETICS offers researchers the ability to quickly and easily build consistent large-scale synthetic natural environments to examine vegetation clutter requirements necessary to accomplish distributed mission training tasks.</p> <p>This dissertation presents and implements the GENETICS algorithm, compares it against other vegetation placement schemes, and outlines how simulationists can use GENETICS to quickly and cheaply build large-scale natural environments. It also touches upon level of detail algorithms, ecotype modeling and how GENETICS can be used to generate land cover data where none exists.</p>				
14. SUBJECT TERMS Terrain Visualization, Vegetation Placement, Virtual Landscapes, Synthetic Natural Environments, Real-time Simulation, Ecotype Modeling			15. NUMBER OF PAGES 169	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**GENERATING ENHANCED NATURAL ENVIRONMENTS AND TERRAIN
FOR INTERACTIVE COMBAT SIMULATIONS (GENETICS)**

William D. Wells
Major, United States Air Force
B.A.E., Georgia Institute of Technology, 1991
M.S.C.S., Air Force Institute of Technology, 1996

Submitted in partial fulfillment of the
requirements for the degree of

**DOCTOR OF PHILOSOPHY IN
MODELING, VIRTUAL ENVIRONMENTS AND SIMULATION**

from the

**NAVAL POSTGRADUATE SCHOOL
September 2005**

Author:

William D. Wells

Approved by:

Dr. Rudolph Darken
Professor of Computer Science
Dissertation Supervisor

Dr. Christian Darken
Professor of Computer Science

Dr. Thomas Lucas
Professor of Operations Research

Dr. Mathias Kölsch
Professor of Computer Science

Dr. Wolfgang Baer
Professor of Information Science

COL (R) Jack Thorpe, Ph.D.

Approved by:

Dr. Rudolph P. Darken, Chair, MOVES Academic Committee

Approved by:

Julie Filizetti, Associate Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Virtual battlefields devoid of vegetation deprive soldiers of valuable training in the critical aspects of terrain tactics and terrain-based situational awareness. Barren landscapes fail to provide trainees with necessary visual cues required to grasp the scale of their surroundings. Without the cover of vegetation, targets are easily visible from the air. Determining line of sight becomes simply a matter of sorting elevations. There is a need to introduce realistic vegetation into our simulators to improve training effectiveness while minimizing the expense typically incurred building such environments.

GENETICS is a new image-based object placement scheme built within Delta3D, the open source simulation engine designed for military training. Using GENETICS, the arduous task of placing vegetation objects across large-scale virtual landscapes is reduced to finding readily-available source data and setting a few parameters. Vegetation-laden terrain is created at runtime without the need for skilled artists, proprietary tools, or playbox-limited databases. The resulting distribution looks realistic and this algorithm can be extended to incorporate a wide variety of environmental factors or to generate geotypical distributions of man-made landscape features. GENETICS offers researchers the ability to quickly and easily construct consistent large-scale synthetic natural environments to examine vegetation clutter requirements necessary to accomplish distributed mission training tasks.

This dissertation presents and implements the GENETICS algorithm, compares it against other vegetation placement schemes, and outlines how simulationists can use GENETICS to quickly and cheaply build large-scale synthetic natural environments. It also touches upon level of detail algorithms, ecotype modeling, and how GENETICS can be used to generate land cover data where none exists.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	MOTIVATION	1
B.	THESIS STATEMENT	3
C.	PROBLEM STATEMENT	4
D.	APPROACH.....	6
E.	SCOPE	8
F.	GOALS.....	10
G.	CONTRIBUTIONS.....	11
H.	DISSERTATION OVERVIEW.....	13
II.	REVIEW OF RELATED WORK.....	15
A.	INTRODUCTION.....	15
B.	CURRENT MILITARY SIMULATION SYSTEMS.....	15
C.	TERRAIN VISUALIZATION SYSTEMS AND DATABASE ACCURACY	16
D.	VEGETATION DATASETS	17
	1. VMAP (Vector Map)	18
	2. LULC (Land Use/Land Cover).....	19
	3. GeoTIFF.....	20
	4. NLCD (National Land Cover Dataset)	20
E.	VEGETATION PLACEMENT WITHIN COMMERCIAL TOOLS	21
F.	MODELING PLANT DISTRIBUTIONS.....	22
	1. Semi-Automated Landscape Feature Extraction.....	23
	2. Ecotope Modeling.....	24
	3. Adaptive Statistical Techniques	26
	4. Plant Growth Simulation	28
	5. Aperiodic Tiling	29
G.	NATURAL ENVIRONMENT RENDERING	31
	1. Billboards & Imposters	31
	2. Point/Line Clouds.....	34
	3. Multiresolution Modeling.....	34
H.	TERRAIN MESH GENERATION	35
	1. Heightmaps.....	35
	2. Fractal Terrain Generation	36
	<i>a. Fault Formation.....</i>	<i>36</i>
	<i>b. Midpoint Displacement.....</i>	<i>36</i>
	<i>c. Particle Deposition.....</i>	<i>37</i>
	<i>d. Fractal Brownian Motion.....</i>	<i>38</i>
	<i>e. Multifractals</i>	<i>39</i>
I.	TERRAIN MESH RENDERING & LEVEL OF DETAIL	39
	1. Geometric Representation of the Surface.....	39
	2. Heightmaps Revisited	40

3.	Level of Detail.....	41
a.	<i>Error Metrics</i>	42
b.	<i>Discrete vs. Continuous Level of Detail (LOD)</i>	43
c.	<i>View-dependent Refinements</i>	44
d.	<i>Geomorphing</i>	44
e.	<i>Occlusion Culling</i>	45
f.	<i>Cracks</i>	45
4.	Quadtrees.....	46
5.	Geomipmapping.....	47
6.	Chunked LOD	48
7.	View Dependent Progressive Meshes	50
8.	ROAM (Real-time Optically Adapting Meshes)	51
9.	SOAR (Stateless One-pass Adaptive Refinement)	52
10.	SOARX.....	56
J.	TERRAIN TEXTURING.....	59
1.	Simple Texture Mapping.....	59
2.	Procedural Texture Generation.....	60
3.	Detail Maps.....	60
4.	Texture Splatting	61
5.	SOARX Texturing	61
III.	METHODOLOGY AND DESIGN	65
A.	OVERVIEW	65
B.	SCENARIOS	65
1.	Mission Planning.....	65
2.	Mission Rehearsal	66
3.	Large-Scale Joint Exercise	67
C.	DESIRED FEATURES	68
D.	NON-GOALS	69
E.	DISCUSSION	69
IV.	IMPLEMENTATION	75
A.	OVERVIEW	75
B.	GATHERING DATA AND CREATING MODEL LIBRARY	75
1.	Organizing Elevation Data.....	75
2.	Locating Imagery and LCC Data.....	76
3.	Building a Vegetation Model Library	78
C.	CREATION AND PARSING CONFIGURATION FILES	78
1.	The soarxterrain.xml System Configuration File	79
2.	The lccdata.xml LCC Configuration File	80
D.	PROCESSING ELEVATION DATA	81
E.	CREATING TOPOGRAPHIC IMAGES.....	82
F.	CREATING LAND COVER PROBABILITY MAPS.....	83
G.	LEVEL-OF-DETAIL AND CULLING	87
H.	AUTOMATIC GENERATION OF LCC DATA.....	88
I.	PROGRAM FLOW AND DEPENDENCIES	90
J.	ALGORITHM OVERVIEW	93
1.	Initialization and Configuration of GENETICS.....	93

2.	Simulation Application Actions That Occur Each Frame	94
3.	SOARXTerrain Class Actions That Occur Each Frame	94
4.	Expansion of the AddVegetation() Method	95
V.	RESULTS	97
A.	OVERVIEW	97
B.	PERFORMANCE	97
C.	CONSISTENCY AND VARIANCE	100
1.	Networked Players	101
D.	ACCURACY	102
E.	RASTER VS. VECTOR DATA	108
F.	COST AND TIMELINESS	111
VI.	SUMMARY, CONCLUSION, AND RESEARCH AGENDA	113
A.	SUMMARY	113
B.	CONCLUSION	115
C.	FUTURE RESEARCH	116
APPENDIX A.	COMMERCIAL IMPLEMENTATIONS	119
	Introduction	119
	3D Nature	119
	Blueberry 3D	120
	Descendor	121
	GScape	122
	OnyxTREE PRO	122
	SpeedTree	123
	Terragen	123
	TreeMagik Pro	124
	Virtual Trees and Foliage	124
	Vue	124
	Xfrog	125
	Other plants	125
APPENDIX B.	CONSISTENCY/VARIATION TEST DATA	127
APPENDIX C.	NETWORKED PLAYERS TEST	131
APPENDIX D.	FUTURE GENETICS RESEARCH	139
	LIST OF REFERENCES	141
	INITIAL DISTRIBUTION LIST	149

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1-1	Flight simulation without surface details or vegetation	1
Figure 1-2	Compiling terrain database instantiations	5
Figure 2-1	Global VMAP1 coverage	18
Figure 2-2	LULC map of Hawaii.....	19
Figure 2-3	Comparison of NLCD (left) and LULC (right).....	20
Figure 2-4	NLCD map of Monterey, CA with legend.....	21
Figure 2-5	Example of a vegetation block	22
Figure 2-6	Comparison of vegetation block vs. sparse distribution of objects.....	22
Figure 2-7	Geospecific tree placement using feature extraction and modeling.....	24
Figure 2-8	Example of ecotope modeling.....	26
Figure 2-9	Example of Voronoi regions	27
Figure 2-10	Voronoi regions before movement of points (big dots) to centroids (small dots) and after using Lloyd's method	27
Figure 2-11	Simulated growth of a self-thinning plant population; red circles die, gold circles are dormant.....	28
Figure 2-12	Simulated plant population of eight different species.....	28
Figure 2-13	Wang Tiles: construction, minimum non-periodic tile set and Wang tiling texture generation.....	29
Figure 2-14	Example of Wang Tiling: input texture, automatically generated set of tiles, & resultant synthesized texture	30
Figure 2-15	Example of view-dependent billboards. Billboard clouds are built for each view-cell. Object distance determines texture resolution & number of billboards needed to represent the shape and appearance of original model...33	33
Figure 2-16	Overhanging terrain	41
Figure 2-17	Example of terrain without LOD	42
Figure 2-18	Quadtree representation of an image.....	46
Figure 2-19	Walking the dependency graph and adding extra vertices where necessary results in a restricted quadtree triangulation	47
Figure 2-20	First three levels of the chunked LOD quadtree	49
Figure 2-21	ROAM triangulation from an overhead view of the terrain.....	51
Figure 2-22	Forced split to avoid T-junctions	52
Figure 2-23	Longest edge bisection.....	53
Figure 2-24	Left: joining four triangles forms a square. Middle: refining triangles individually results in cracks. Right: crack prevention.....	54
Figure 2-25	Bounding sphere hierarchy	55
Figure 2-26	Error is projected from the nearest point on the bounding sphere.	55
Figure 2-27	Left: The column in the front is inside the viewing frustum. Right: Tilting the camera a bit results in the top vertex of the column being culled by the original algorithm, making the whole column disappear.....	57
Figure 2-28	Culling the bounding spheres against the view frustum. The shaded area indicates the inner side of the frustum.	58
Figure 2-29	Left: embedding detail map Right: bi-linear interpolation	62

Figure 2-30 Comparison of surface combining techniques (base, displacement, addition) ..	63
Figure 3-1 LOD and aerial perspective (for tank, helicopter, & airplane)	71
Figure 4-1 DTED™ directory structure	76
Figure 4-2 CalView path/row numbering scheme and image from path 43 row 035.....	77
Figure 4-3 USGS Seamless Data Distribution Server	77
Figure 4-4 OpenFlight® LOD model created with REALnat®	78
Figure 4-5 Height map, relative elevation map, and slope map with aspect angle.....	82
Figure 4-6 Third nearest neighbor weighting scheme	83
Figure 4-7 Picked points vs. smoothed with masking	84
Figure 4-8 Evergreen probability map	85
Figure 4-9 Examples of GENETICS vegetation placement with placeholder objects	86
Figure 4-10 Examples of GENETICS vegetation placement with commercial objects.....	87
Figure 4-11 Example of man-made object distribution	87
Figure 4-12 LCC training data shown in the upper left corner. The rest is false color elevation data.	89
Figure 4-13 Maximum likelihood estimate of the LCC type with the k-nearest neighbors estimator.....	90
Figure 4-14 Delta3D's underlying open source libraries.....	91
Figure 4-15 GENETICS “black box” diagram	92
Figure 4-16 GENETICS quadtree structure.....	96
Figure 5-1 Scene 1 with random number seeds 42, 47, and 12.	101
Figure 5-2 GENETICS networked player test	101
Figure 5-3 Google Maps imagery vs. GENETICS vegetation distribution of triangular vegetation growth near Watsonville, CA.....	103
Figure 5-4 NASA's World Wind (left) vs. GENETICS (right).....	104
Figure 5-5 Google Earth image of Aromas, CA eucalyptus grove (note 25m and 100m lines).....	105
Figure 5-6 High random distribution with and without topographic influences	106
Figure 5-7 Low random distribution with and without topographic influences	106
Figure 5-8 Top: GENETICS high with 2 looks per pixel; Middle: GENETICS high with 1 look per pixel; Bottom: strict high resolution LCC placement and imagery	107
Figure 5-9 Top: GENETICS low with 20 looks per pixel; Bottom: strict low resolution LCC placement and imagery	108
Figure 5-10 Comparison of LULC data (left) vs. NLCD data (right)	109
Figure 5-11 Aerial imagery of eucalyptus grove	109
Figure 5-12 Comparison of LULC (left) vs. NLCD (right) vegetation distributions with LCC-based (top) and imagery-based (bottom) ground textures	110
Figure 6-1 Training requirement/visual fidelity matrix research process.....	118
Figure C-1 Tank behind the tree test (with and without the tree).....	131
Figure C-2 Tank behind the tree test – Part 2	132
Figure C-3 Searching for the tank.....	133
Figure C-4 Easy to see tank; hard to see helicopter.....	134
Figure C-5 Tank blending in with bushes.....	135
Figure C-6 Tank behind a dead bush	136

LIST OF TABLES

Table 2-1 Variables used to calculate vegetation placement (From Hammes)	25
Table 5-1 GENETICS Runtime Performance.....	98
Table 5-2 GENETICS Viewport Objects	99

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS, ACRONYMS, SYMBOLS

ABN	Airborne
CGF	Computer Generated Forces
CLOD	Continuous Level of Detail
CORINE	Coordination of Information of the Environment
CPU	Central Processing Unit
DEM	Digital Elevation Model
DFAD	Digital Feature Analysis Data
DoD	Department of Defense
DTED	Digital Terrain Elevation Data
fBm	Fractal Brownian Motion
fps	frames per second
GENETICS	Generating Enhanced Natural Environments and Terrain for Interactive Combat Simulations
GEOTIFF	Geographic Tagged-Image File Format
GIS	Geographic Information System
GPU	Graphics Processing Unit
HLA	High Level Architecture
JFCOM	Joint Forces Command
JOG	Joint Operations Graphics
LCC	Land Cover Classification
LIDAR	Light Intensity Detection and Ranging (a.k.a. LADAR)
LOD	Level of Detail
LULC	Land Use Land Cover
M&S	Modeling and Simulation
MOUT	Military Operations in Urban Terrain
MOVES	Modeling, Virtual Environments and Simulation
NGA	National Geospatial-Intelligence Agency
NLCD	National Land Cover Dataset
NPS	Naval Postgraduate School
OPFOR	Opposition Force
PRNG	Pseudo Random Number Generator
RGB	Red, Green, Blue (a.k.a. a raster image)
ROAM	Real-time Optically Adapting Meshes
RTI	Runtime Infrastructure
SA	Situational Awareness
SIGGRAPH	Association for Computing Machinery's (ACM's) Special Interest Group on Computer Graphics
SME	Subject Matter Expert
SOAR	Stateless, One-pass Adaptive Refinement
SOARX	SOAR Extended
TEC	Topographic Engineering Center
TIN	Triangulated Irregular Networks

USGS
VMAP
XML

United State Geographic Survey
Vector Map
Extensible Mark-up Language

DEDICATION

For Kathi...

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Current synthetic terrains are bleak, desolate places that share a strong measure of commonality with desert regions. However, seeing that the majority of the Earth's landmass is not desert, such terrain is not representative of the actual environment that today's soldiers will likely find themselves in in the future. One of the ongoing problems with a relatively featureless synthetic environment is one's inability to grasp the scale of the terrain. It is nearly impossible to determine distances or speed in a world devoid of a single bush, tree, or surface detail necessary to establish depth cues. (Peitso, 2002; G. T. Wright, 2000) Large polygonal meshes draped with blurry satellite imagery (see Figure 1-1) are almost entirely absent of any visual cues to aid the infantry soldier on the ground or the low flying pilot. To improve simulator-based training, this situation needs to change. Plausible surface details must be added to both the geometry and the textures covering that geometry. Vegetation representative of the type and density found within the actual environment should exist to build further depth cues, but more importantly, to make the training more difficult, especially in the realm of target detection.



Figure 1-1 Flight simulation without surface details or vegetation

Within today's cockpit simulators, it is far too easy for pilots to quickly “find, fix or track, and target anything that moves on the surface of the Earth.”¹ When the only object protruding from the terrain surface is an enemy tank with no cover to hide behind, the task of acquiring and destroying your enemy, while never easy, is greatly simplified. In an era of US forces “tank-plinking” Iraqi armor units in the desert, it is easy to rationalize that this is a valid training environment. However, let’s think about how a simulator-trained pilot of today would perform if taken back several decades ago to Vietnam. The jungles are dense and where there aren’t trees, there are rice paddies and swamps. Ground combatants often find themselves locked in close quarters battles. From over ten thousand feet in the air, the battlefield is a thick ocean of green. It is nearly impossible to tell friendly from enemy forces. Even if the location of an enemy force is known, the ability to accurately pinpoint a particular target and destroy it on a first pass is highly questionable. Fratricide becomes a real possibility in such situations. Adding vegetation to synthetic battlespaces improves realism by forcing trainees to deal with increased visual clutter in their environment.

Naturally, such terrain characteristics are highly desirable, and thus for detailed simulated environments like those found within the America’s Army® game (Davis, 2003; Zyda, Mayberry, Wardynski, Shilling, & Davis, 2003), a team of artists is hired to handcraft custom terrain databases. These databases are not only simulation system specific (e.g. America’s Army was constrained to using .25 by .5 km terrains by the Unreal® 2 game engine), limiting their reusability or interoperability with other simulations, but they take a great deal of time to create. Additionally, such databases are typically bounded by a player’s expected actions and viewpoints. If players deviate from the level designer’s expectations, they quickly discover places within the world that simply do not “exist” (where early mapmakers might have placed the warning “Here there be dragons!”). These limitations prevent simulation scalability throughout the full spectrum of military operations and this is part of the reason that user-controlled vehicles have not been introduced into America’s Army®. A helicopter, aircraft or a fast-moving ground vehicle would quickly discover the boundaries of the tiny America’s Army®

¹ Remarks by Air Force Chief of Staff Gen Ronald R. Fogleman at the Air Force Assoc. Symposium, Los Angeles, Oct. 18, 1996.

terrain box. Another drawback is that these static terrain databases must be published and distributed throughout the simulation network. Even small changes to the terrain database would likely require a sizeable download by each client system (or a CD or DVD installation).

We have designed a system to overcome these traditional terrain database limitations by automatically generating detailed landscapes at runtime from a common set of readily-available source data that guarantees the same terrain environment is created by all the hosts within a heterogeneous simulation network. With the simple change of a random number seed, a new terrain with similar landscape characteristics can be generated without the requirement of a team of artists to manipulate a proprietary database. This feature allows trainers the flexibility to use the same terrain repeatedly or use a new one each time, forcing trainees to avoid dependence upon the static nature of most simulation databases.

B. THESIS STATEMENT

The objective of this research is to generate enhanced natural environments and terrain for interactive combat simulations (GENETICS) as demanded by specific training requirements. Our long-term research goal is to automatically replace the barren landscapes found within most 3D combat simulations with detailed terrain and natural surroundings that dramatically increase both the realism and difficulty of the training environment. This dissertation shows how one can develop such terrain models in significantly less time and at a higher level of visual detail than previously attainable with a traditional terrain database creation approach.

This dissertation concerns the design and implementation of an automated vegetation distribution algorithm within a distributed virtual environment. We show that given a region's elevation data and corresponding land cover classification information, we can algorithmically place ecotypic vegetation objects (e.g. trees, bushes, etc.) to produce consistent, realistic landscapes where topographic features impact vegetation placement and natural blending occurs between ecotype regions.

C. PROBLEM STATEMENT

Current terrain rendering engines within Department of Defense (DoD) simulators use elevation and cultural feature data that are preprocessed and optimized in a semi-automated fashion by commercial systems (and their highly-trained operators) and saved into proprietary formats. Creating terrain databases is a time-consuming and expensive process and thus once an acceptable model of the environment has been built, there is a temptation to use it as long as possible in a wide a variety of applications. Thus, a five year old model of Fort Irwin may stand-in as a suitable training environment for present-day Iraq or Afghanistan. But what is a suitable stand-in for Indonesia? With so much information being processed by remote-sensing devices, why do we even need stand-in databases when current data of the desired location is available? The question revolves around the time, money, and skill/tools needed to create a terrain database.

Let's say that we are going to create a "current" terrain database of the Korean peninsula (using a mosaic of various data elements dating back a decade) for the annual large-scale Ulchi Focus Lens exercise. Over the course of a year, the massive database is finally completed and distributed out to sites in Korea, Japan, and the United States. These sites all have specific needs (e.g. performance requirements, interest areas, resource limitations, etc.) that will force the terrain database modelers to generate "lesser" instantiations of the "true" terrain database for their customers (see Figure 1-2). Due to the cost and manpower required to create the master database, it will be used for at least the next five years. Minor annual updates may be generated to resynchronize the database between units as organizations lose, tweak, or repurpose their portion of the exercise database for use in other activities like mission planning and analysis. Without the tools or skill set needed to update their instantiation of the database, organizations must wait for any database updates to flow to them. But what if an organization, with access to a common repository of source data, could create their own synthetic environment from the latest data, tailored to their own needs, that didn't require the intermediary step of having someone create and maintain a master database and then publish a specific instantiation for their application? Such an approach could be useful across a wide variety of geospatial applications (training, mission planning, GIS, etc.).

When synchronization is needed with other units, the sharing of a small set of parameters and the commonality of the source data serve to create a coherent environment without the need to publish and distribute multiple versions of the terrain database. As requirements, computing resources, and source data change, so can the terrain representation, independent of a static database.

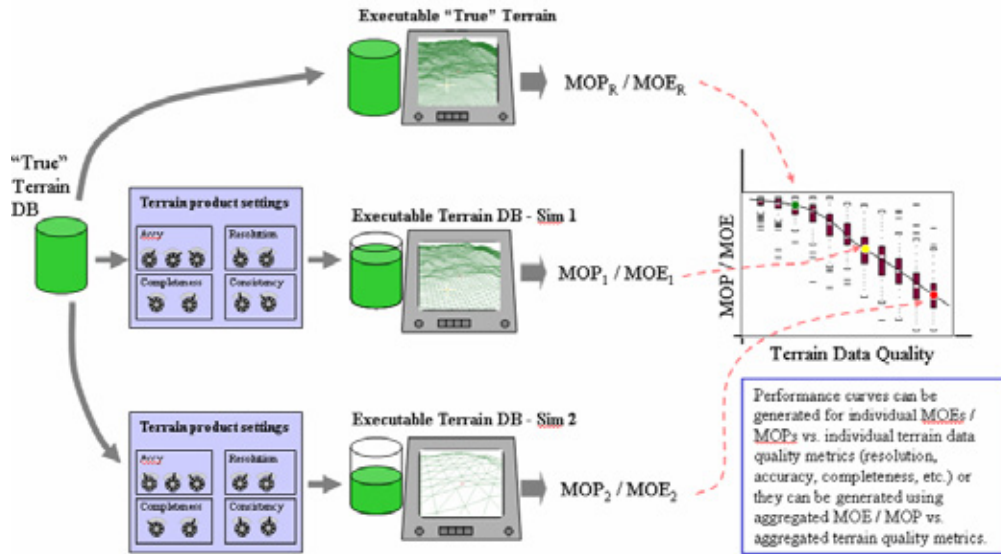


Figure 1-2 Compiling terrain database instantiations
(After Wright, 2005)

Generating simulated environments in this manner moves terrain database generation from a “push” technology (i.e. waiting on a central site to send data; hopefully in a useable format) to a “pull” technology (i.e. taking data from a site or sites and fusing this information into a desired product). “Push” is akin to receiving a yearly set of encyclopedias while “pull” is how people look up information on the Internet. While encyclopedias are authoritative resources, they are expensive, static, may miss items of interest to you (especially in the realm of current events), focus on items that are irrelevant to your study, and are formatted in a particular set style (e.g. World Book® vs. Encyclopaedia Britannica®). The Internet is an inexpensive, dynamic, non-authoritative fount of information (although authoritative sites do exist and can be searched for

content). Current events are a particular specialty of the Internet (sometimes to the point of neglecting old content). The Internet user can focus a search on specific key terms, get back various reports on that subject, and then must internally deconflict and fuse the data to decide on how to act upon that information.

The challenge of GENETICS is to, without proprietary tools or terrain database preparation, pull specific raw data (elevation, imagery, and land cover) from a repository, fuse this data in a procedural manner to enhance its apparent quality, and add vegetation objects to the scene that conform to the terrain in a manner that is similar to the arrangement within the actual environment. All of these actions should occur automatically during the execution of the simulation system. We make the assumption that land use or land cover classification data exists to describe the placement of vegetation in general terms. Due to the dynamic loading and processing of the source data, players are not limited by the physical boundaries imposed by traditional preprocessed terrain databases (i.e. no “playbox” restrictions). Where data is missing or incomplete, our system can extrapolate missing land cover data. By using the same source data, configuration parameters, and placement algorithm, all players within the distributed virtual environment can generate identical representations of the battlespace.

D. APPROACH

It is the contention put forward in this dissertation that the generation of synthetic terrains can be quick, automatic, based on raw source data, consistent between players, and responsive to the requirements of the training audience. Believable landscapes (i.e. representative of the actual environment) can be created easily by novice users without training in geographic information systems (GIS) or other terrain creation tools. In order to design, construct, and test this concept, we:

1. Design a system based on functional specifications identifying typical users of GENETICS and how they would interact with the system (e.g. gathering source data, setting parameters, etc.) to support their desired training objectives and/or terrain visualization requirements.

2. Identify suitable continuous level of detail algorithms. Compare and contrast these algorithms to determine suitability with respect to implementing an object placement algorithm and ability to increase the apparent resolution of source data using a noise function.
3. Implement the selected continuous level of detail algorithm to create a terrain mesh and drape imagery over the mesh with noise-generated details added to improve the apparent quality of the source imagery.
4. Identify suitable land cover classification datasets. Compare and contrast these datasets to determine suitability with respect to an object placement algorithm and user expectations of terrain geospecificity.
5. Develop a vegetation object placement algorithm based on the processing of raw and derived land cover classification and elevation data. Placement must be automatic, repeatable, and guaranteed to put the same size and type of object in the same location and orientation given the same input data and configuration parameters (e.g. random number seeds). Address the ability of objects to “collide” or overlap with other objects.
6. Examine level of detail techniques to realistically represent vegetation objects with reduced geometric complexity. Use a level of detail (LOD) technique to construct vegetation objects and spatially organize these objects within our system to improve rendering performance.
7. Test the GENETICS framework within a networked environment to demonstrate consistency of object placement. Compare object placement vs. satellite and aerial imagery of vegetation cover to examine the geotypical and geospecific aspects of the algorithm. Compare GENETICS object placement against traditional techniques.

It should be noted that large-scale vegetation placement is one of the most significant missing elements within today’s simulation systems. It is an area where source code and academic papers are either not publicly available or simply don’t exist. Fortunately, vegetation generation can be examined somewhat independently of other terrain visualization components. While elevation data is needed to determine topological features and their influence on ecotope properties, this data (and its resulting mesh) need not be optimized with a continuous level of detail scheme nor noise-enhanced. Similarly, surface textures, lighting and shadows, and other terrain visualization elements can be studied and implemented independently from the vegetation placement algorithm.

E. SCOPE

Areas considered out-of-bounds for this dissertation effort included dynamic terrain, overhangs, collision detection with vegetation objects, triangulated irregular networks, road networks, automated feature extraction, urban terrain, linkages to constructive simulations and computer generated forces (CGF), populating building interiors, and dynamic environmental effects.

Dynamic terrain requires the management of a dynamic shared state and consequently a networking scheme capable of keeping existing players up-to-date on state changes within the environment and to initialize the current state of the world for late joiners. (Singhal & Zyda, 1999) While dynamic terrain is a worthwhile goal, it is also a classic networked virtual environments problem that has many solutions in use today. The GENETICS system is not expected to greatly help or hinder the process of exchanging terrain state changes between networked players.

Overhangs, caves, tunnels, and other terrain features that are not well represented within a regular grid-based (e.g. elevation postings) layout will not be considered since our system is designed to automatically enhance existing measured elevation data. Certainly, techniques have been created that allow for 3D terrain measurements and volumetric rendering options are available to accurately display this data. Unfortunately, the vast majority of military terrain databases do not contain the required information to support volumetric rendering. In the end, since our source data doesn't depict overhangs, neither will GENETICS.

Collision detection with vegetation objects is a capability that will only be addressed from the perspective of object placement (e.g. preventing intersecting tree trunks). Players colliding with vegetation objects was not intended to be a focus of this research. Readers interested in such matters are invited to examine Ming Lin's survey paper on real-time collision detection or Gino van den Bergen's text. (Lin & Gottschalk, 1998; van den Bergen, 2003)

Triangulated irregular networks (TINs), a method of reducing the polygonal complexity of elevation meshes, are ill-suited for procedurally refined continuous level of detail algorithms. Further explanation for scoping out TINs is given in the next chapter.

Road networks are a hallmark of TIN terrain databases since TINs allow for road networks to be “cut-in” to the elevation mesh. This technique cannot be applied to grid-based elevation meshes. Additionally, to properly visualize a complex road network requires the generation of highway overpasses and under-crossings. This in-turn requires the use of multiple elevation surfaces which has already been discounted in the “overhang” paragraph above.

Automated and semi-automated feature extraction of vegetation is a desirable trait of high fidelity terrain visualization systems (e.g. for military operations in urban terrain (MOUT)). Work has already been accomplished in this field by Georgia Tech’s Wasilewski et al., University of Zurich’s Hirtz et. al, and the Naval Postgraduate School’s Baer and Campbell. (Baer & Campbell, 2003; Hirtz, Hoffmann, & Nüesch, 1999; Wasilewski, Faust, Grimes, & Ribarsky, 2002) Our intent is not to replicate geospecific object placement within a particular locale using hard-to-find high resolution data (e.g. LIDAR elevation data and sub-meter imagery), but to use readily available medium-resolution elevation and imagery data that can be used to create large-scale geotypical terrains. Additionally, high resolution datasets are ill-suited to generating large-scale landscapes due to the tremendous storage requirement. A geocell (i.e. 1 degree x 1 degree; nominally 100 km x 100 km) of LIDAR data sampled at 30 cm would require over 111 billion elevation postings. Using 8-byte floats for the x, y, and z values would require 2.4 terabytes of storage. Similarly, using six inch resolution imagery data (per the Georgia Tech study) would result in a 656168 x 656168 pixel image for our geocell. When stored in a lossless geospatial raster format (e.g. GeoTIFF) at 3 bytes per pixel, this massive image would require another 1.2 terabytes of storage.

Urban terrain generation (either procedural or based on geospecific imagery) is a well established research area (Frère, Vandekerckhove, Moons, & Van Gool, 1998; Frueh & Zakhor, 2003; Greuter, Parker, Stewart, & Leach, 2003; Haala & Brenner, 1999; Parish & Müller, 2001; Takase, Sho, Sone, & Shimiya, 2001; Vandekerckhove, Frère,

Moons, & Van Gool, 1998) and it was not our intent to duplicate this work. Such urban generation systems should be able to work in cooperation with GENETICS. For example, GENETICS adds terrain and vegetation to the scene graph, the “Urban Generator” software adds buildings and roads, and the host simulation adds moving entities through the reception and processing of entity-state packets. In a simpler case, the urban environment is masked-out by the GENETICS user, preventing the vegetation placement algorithm from operating within the masked-out region. This allows a high-resolution model of an urban scenario to be geospecifically positioned within the context of the larger GENETICS-created environment.

Linkages to constructive simulations and CGFs will be dependent on a host’s particular simulation application and not directly tied to the GENETICS terrain visualization system itself. Thus, if a host simulation system supports the reception of entity-state packets from another simulation, GENETICS will not prevent those entity objects from being added to the host simulation’s scene graph. However, without collision detection, graphic anomalies are likely to occur (e.g. a tank can drive through a tree without damage to the tree or the tank).

While portions of the vegetation placement algorithm (particular the ecotyping routines) could be extended for use to help populate building interiors, it is not our intent to add or prove this capability within an existing urban terrain generator or visualization system.

Dynamic environmental effects (e.g. weather, floods, earthquakes, fire, etc) are important aspects within some terrain visualization systems, but will not be addressed within this dissertation research. As these effects would require modifications to the terrain’s ecotyping during run-time, these effects fall into the category of dynamic terrain and thus the realistic handling of such effects is simply beyond the scope of this project.

F. GOALS

The overall goal of this research is to use our terrain visualization software within a wide variety of combat simulators (e.g. dismounted infantry, ground vehicle, helicopter, aircraft), including those that require differing LOD requirements. The terrain displayed

by each player will be perceptually the same as determined by the LOD requirements of the given entity. Thus, a detailed geometric tree object seen by an infantry soldier may be represented as a flat billboard to an aircraft entity, but both will see the identical tree placement and general object characteristics (e.g. size, shape, color, density, etc). GENETICS is responsible for feeding identical scene graphs to each player's image generator (IG), which will then determine how the scene is actually rendered.

For networked simulations, the GENETICS software package and raw source data used to create the terrain must be common across all networked players. The emphasis here is to guarantee that given identical matched-set source data (i.e. corresponding georeferenced elevation and land cover datasets), GENETICS will automatically create the same realistic natural environment for each player. This terrain can be recreated by keeping or reusing its corresponding configuration parameters. By changing these parameters within a simple text file, the same source data can be used to create differing terrains with an aggregate similarity to previous runs, but with differences in the details.

The long-term goal of this research is to examine the potential set of simulator-based training tasks for missing visual cues, to automatically construct a believable landscape using GENETICS to satisfy missing visual cues, and to demonstrate a resulting improvement to simulator-based training. This research agenda is outlined in Chapter VI.

G. CONTRIBUTIONS

While previously it has taken teams of artists to create static, small scale, custom-tailored landscapes, our approach generates large-scale realistic terrains for any place on Earth where elevation and land cover data exists or any virtual place where the same input data is available. Terrain can be reused or regenerated afresh with new parameters in response to the needs of the training audience. All of these actions (i.e. elevation mesh creation, optimization, and vegetation placement) take place automatically without the additional time and expense needed for skilled artists or proprietary tools. Variation in the terrain can be assured by randomness and noise functions, but ecotope properties are used to help prevent improbable situations (e.g. trees growing in bodies of water) while allowing the flexibility for terrains where those possibilities may exist (e.g. Louisiana

bayous). With a minimum amount of source data and parameters, terrains can be synchronized between clients easily.

With regards to this dissertation's selection of implementing vegetation generation, we have these words from the National Geospatial-Intelligence Agency's Kurt Hoglund, "NGA is most interested in a methodology that represents geotypical data in a 'natural' fashion that does not involve tiling nor repeating of object types in a regularized fashion. Being able to handle such things as vegetation in a geotypical, yet stochastically placed set of objects for representation based on geographic placement is of high interest not only to NGA, but the M&S community and the DoD and IC (International Coalition) operational communities." (Hoglund, 2004) Clearly, there existed an unmet need to automatically generate geotypical terrain and vegetation from geographic source data. GENETICS has solved this problem.

GENETICS also marks a philosophical shift in the design and creation of virtual landscapes. No longer must master terrain databases be constructed through the use of commercial tools and then published into an executable format for the needs of each simulation application. The time, expense, and expertise required to generate these databases and their static products prevents them from being responsive to changing source data, processing power, or player requirements. GENETICS automatically processes source data and produces terrains tailored for each simulation application. The desired level of terrain consistency between heterogeneous simulations will force necessary agreements in determining configuration parameters, but the terrain created by each simulation can be generated from today's source data versus a prepared master database built months or years ago. This flexibility allows one to immediately visualize a region (given that data exists) without prior warning. The upshot is that the GENETICS philosophy of terrain creation becomes a key enabler for instantaneous scenario generation in response to pop-up crises for crisis action planning, mission rehearsal, or quick-turn analysis. The traditional approach of publishing simulation-specific static terrain models from a master database based on stale source data is now archaic.

A byproduct of this work is an accounting and comparison of major terrain algorithms, plant community representations, and terrain enhancement schemes known to

the author. These methodologies are described in the following chapter. This terrain visualization research also explores the use of programmable graphics hardware for improving the performance and visual appeal of the natural environment. The lessons learned from our experiences with this technology will hopefully serve to drive further development in the creation of realistic natural environments using these new resources.

The long-term practical benefit of this work is to improve tactical training by giving players a more realistic environment in which to operate. Even a high fidelity helicopter simulator is only a part-task trainer if it operates within a restricted unrealistic environment. It is possible to have simulators and simulated forces engage in a multi-spectrum tactical conflict where the natural environment takes on an active role in the experience and is no longer simply a backdrop. It is only at this point, when the ground looks real and foliage hides your view of the enemy, that terrain can truly work towards becoming a fully-fledged entity within the distributed virtual environment.

H. DISSERTATION OVERVIEW

The remainder of the dissertation is organized as follows: Chapter II provides a detailed description of previous work, current research, and background material relevant to this dissertation. Chapter III conceptualizes the functional and technical design of GENETICS. Implementation of the GENETICS design is presented in Chapter IV. Chapter V demonstrates and discusses test results conducted using the GENETICS framework. The dissertation concludes with Chapter VI which summarizes findings, provides recommended practices for designing terrain visualization systems, and outlines a research agenda for testing training improvement using GENETICS. The dissertation's appendices provide additional material from the research studies and algorithm refinement process, and descriptions of major commercial vegetation tools.

THIS PAGE INTENTIONALLY LEFT BLANK

II. REVIEW OF RELATED WORK

A. INTRODUCTION

Although there is no single source for information on terrain visualization algorithms & implementations, one of the best on-line clearinghouses is the Virtual Terrain Project (www.vterrain.org) that keeps track of most of the latest advances in terrain visualization techniques to include rendering & vegetation algorithms. While providing modest critique and commentary, it offers hundreds of links to a wide assortment of published papers, implementations, and sources of data. Nearly all of the concepts discussed below can be found there. But before we talk about the various techniques that go into creating an optimal terrain visualization system, it might be best to discuss what is seen within current military simulation systems.

B. CURRENT MILITARY SIMULATION SYSTEMS

Today's platform simulators (e.g. tanks, helicopters, aircraft, and soldier stations) generally rely on expensive third party tools like TerreX's Terra Vista™ (\$35K for the application, \$6K in annual fees, and \$5K for training). Terra Vista™ creates simulator-optimized terrain databases that can either be exported using a standard file format (e.g. Multigen-Paradigm's OpenFlight®) or a high-performance proprietary format (TerreX's TerraPage™). Terra Vista™, like most commercial terrain database tools, imports a variety of source data (e.g. elevation, imagery, vector products, etc.) that can be layered with cultural feature data (e.g. buildings, power lines, road networks, etc.) through the placement of 3D models on the terrain surface. Terrain elevation data is often turned into a series of static level of detail (LOD) meshes using Triangular Irregular Networks (TINs) in a method similar to the Chunked LOD technique that will be discussed later. No geomorphing occurs, so the terrain "pops" as it transitions between discrete LODs. Fading between LODs must be done within the simulator application to reduce this popping effect since Terra Vista™ only creates the terrain database and is not responsible for visualizing it outside of its own database creation and modification application. Terra Vista™ can manage relatively large databases and is able to merge various resolutions of

source data, but is unable to cope with missing data or create detail where that detail is lacking. There is also the issue of handling large terrain databases where static LODs are used since a substantial amount of data must be redundantly stored on the hard disk for each LOD resolution. Since Terra Vista™ supports terrain paging, terrain databases are typically cut into many tiles. Imagine a terrain that is cut into 6 tiles at the coarsest level of detail. Using a quadtree hierarchy to manage LODs means that the second LOD will require 24 tiles. The third and most detailed LOD will divide the terrain into 96 tiles. If each tile is stored as its own file (or more likely a series of files: mesh, textures, object placement, etc.), this means that 126 separate files must be stored just to visualize the terrain surface and a single location in this terrain database is stored within 3 separate files.

TINs, while a great way of reducing polygons, are generally incompatible with the notion of continuous levels of detail (CLOD) due to the connectivity information required to create and maintain the TIN. (Luebke, 2003) Grid-based CLOD methods maintain a record of source elevation data so that triangles can be decimated or tessellated dependent on the camera's position, speed, and view frustum. For Terra Vista™, the source data is used to create the terrain database, but is not stored within the terrain database itself. Thus, if the database modeler wishes to change the visual resolution of the terrain database (e.g. based upon the visualization needs of each platform simulator), the database must be reconstructed within Terra Vista™ using the new parameters and exported out to numerous files using the specified file format.

C. TERRAIN VISUALIZATION SYSTEMS AND DATABASE ACCURACY

Geographic Information Systems (GIS), while typically not a major component of simulator-based training, offer valuable insights into the display of large terrain datasets. In the GIS world, a critical feature of the terrain visualization systems lies in the accurate portrayal of the dataset. It is precisely because of this intense desire for accurate data that we discover that terrain datasets are inherently inaccurate. Due to the irregular curvature of the Earth, we cannot accurately display terrestrial datasets in 2-dimensions (e.g. images and maps). Thus, a wide range of projections, coordinate systems, and datum

planes were created to minimize distortions within particular regions of interest. As we travel away from these reference points, the data (and thus our image) becomes increasingly distorted. Within a particular area of interest, we can generally be assured of a reasonable level of accuracy. All terrain data has a personality and pedigree associated with it and there exists no one true, perfect dataset. “Thus, no representation is likewise true or perfect, all contain errors, omission, abstractions, simplifications, points-of-view, and exaggerations, and that far from damning all representations as flawed, this is exactly what makes them valuable and useful, but to be carefully chosen amongst and deployed.”(Ervin & Hasbrouck, 2001) Ervin and Hasbrouck also note that one of the challenges in creating 3D landscape representations is that no one dataset truly captures the dimension of time and change. A terrain surface created from a single satellite image doesn’t reflect seasonal change or that the following year a wildfire consumed the entire forest. Days-old data fails to capture overnight weather events. Conversely, immediacy of data is not always the preferred solution. Capturing dramatic (e.g. wildfire, flood, etc) or subtle (e.g. clouds, seasons) natural events may not benefit future data consumers searching for “typical” representative landscape data of a region. It is precisely for this reason that the cloudless 1997 Earth composite known as the GeoSphere by Tom Van Sant took two years to create. Increased accuracy also comes with its own host of problems. LIDAR (Light Intensity Detection and Ranging), a laser-based scanner that can be used on an airplane or satellite, can capture elevation data points down to 1cm in resolution (effectively capturing vegetation canopies and the underlying terrain). Of course, storage and processing of this amount of data makes it impractical for large-scale terrain visualization (as discussed in Chapter I).

D. VEGETATION DATASETS

Realistic vegetation placement requires some amount of real-world landscape ecology data which is typically stored in either vector or raster data structures. Vector data includes points, lines/arcs, and polygonal areas linked to a corresponding set of attribute data. Raster data uses regular grids composed of values commonly linked to a legend of attribute data. Early GIS users favored vector data for its hard disk space

saving ability despite the labor-intensive nature to generate such data vs. the automated image-processing techniques that could be applied to create raster images. Today's powerful computers, massive hard drives, and high resolution images have all but eliminated the advantages of using vector maps for storing land cover classification data. Our approach uses raster data while today's commercial packages rely on one or both of two popular vector datasets: VMAP/DFAD and LULC.

1. VMAP (Vector Map)

VMAP is a National Geospatial-Intelligence Agency (NGA) vector/polygonal dataset often used by the U.S. military. VMAP is based on vectorized versions of NGA's Joint Operations Graphics (JOGs) paper maps. VMAP is a replacement for the older DFAD (Digital Feature Analysis Data) format. VMAP Level 1, the most readily-available data (see Figure 2-1 for coverage), has a scale of 1:250,000 and a circular error of approximately 125 meters. VMAPs are organized into 10 thematic layers where each layer is stored as a single coverage with point, line, area, and text designations. For example, a vegetation layer can consist of points (e.g. oasis), lines (e.g. firebreaks and tree-lines), areas (e.g. crops, grass, orchard, swamp, forest, tundra, and vegetation voids), and text labels. This data is not directly viewable or editable without proprietary tools such as Manifold®, ESRI's ArcView™, or Multigen-Paradigm's Creator Terrain Studio™.

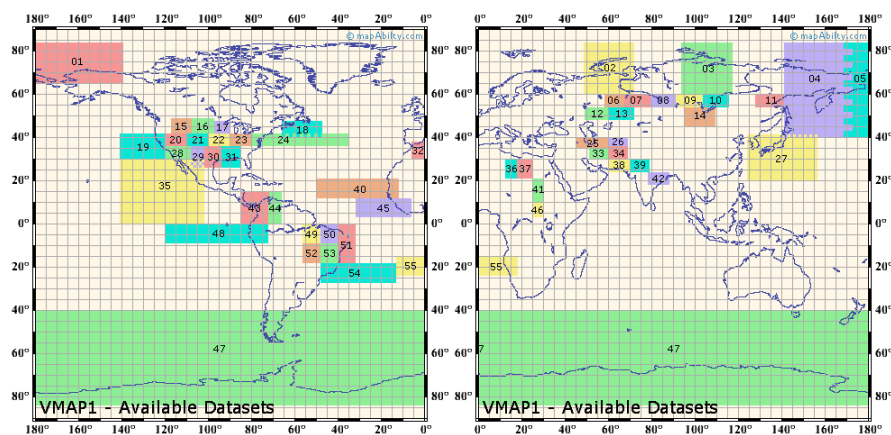


Figure 2-1 Global VMAP1 coverage

2. LULC (Land Use/Land Cover)

LULC (Land Use/Land Cover) is a U.S. Geologic Survey (USGS) vector/polygonal dataset consisting of historical land classification data based on manual interpretation of 1970's and 1980's aerial photography, land use maps and surveys. There are 21 categories of cover type. LULC data is available for conterminous U.S. and Hawaii (see Figure 2-2), but coverage is incomplete. LULC data is based on 1:100,000 (only available for Hawaii and nine other states) and 1:250,000-scale USGS topographic quadrangles. LULC's spatial resolution depends on the feature types represented. The minimum polygonal area for non-urban or natural features at 1:250,000-scale is 16 hectares (40 acres) with a minimum width of 400 meters (1320 feet). For urban and built-up classification types, the minimum resolution is 4 hectares and 200 meters. The native LULC format, GIRAS (Geographic Information Retrieval and Analysis System), is readable by commercial tools such as TerraTools®, Terra Vista™, Manifold® and ArcInfo™. Like VMAP, LULC data is not directly viewable using a web browser or image viewing software and therefore must be sampled and converted into a grid-based format (e.g. CTG, Composite Theme Grid) to produce raster images.

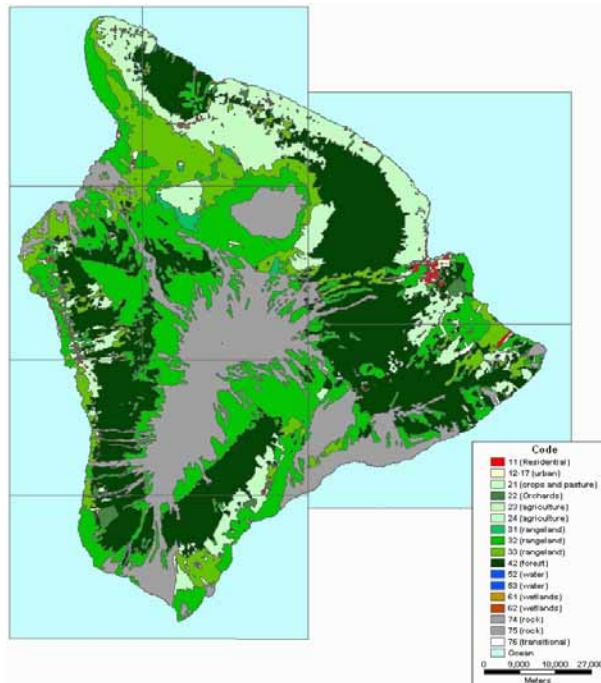


Figure 2-2 LULC map of Hawaii

3. GeoTIFF

For vegetation placement, we use land cover classification (LCC) GeoTIFF images of our region of interest. GeoTIFF (Geographic Tagged-Image File Format) is a popular raster file format created by a consortium of cartographic and surveying organizations to establish a TIFF-based interchange representation for georeferenced raster imagery. GeoTIFF imagery can originate from satellite imaging, aerial photography, scanned maps, digital elevation models, or from geographic analyses tied to a known model space or map projection. GeoTIFF images use a small set of reserved TIFF metatags to store georeferencing information, such as the projection type, reference datum plane, coordinates of the four corners of the image, and the “physical size” of each pixel. Georeferenced GeoTIFF images allow one to match and compare identical pixel regions within multiple images. As an image format, GeoTIFFs are viewable by most image viewing software and readily useable in visualization applications as texture maps. Figure 2-3 demonstrates the resolution disparity between high resolution raster (30m) and vector (1:100K) LCC data.

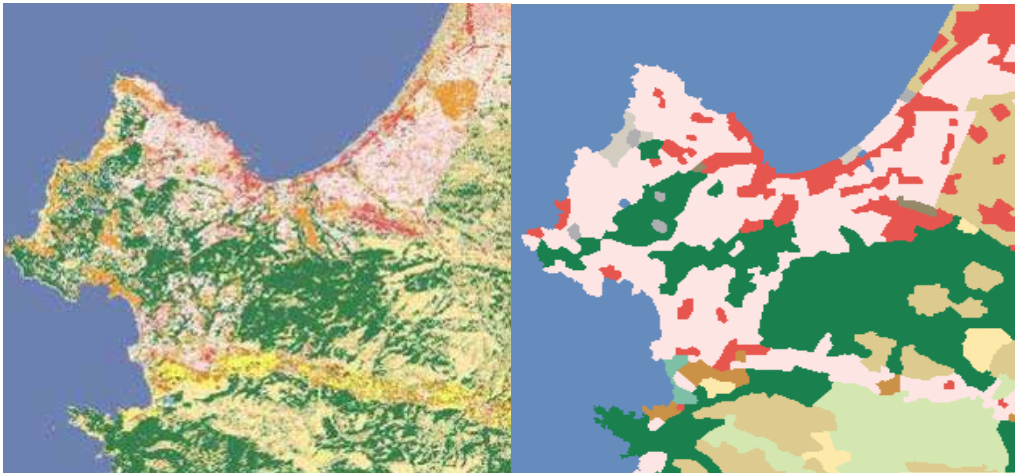


Figure 2-3 Comparison of NLCD (left) and LULC (right)

4. NLCD (National Land Cover Dataset)

The National Land Cover Dataset (NLCD) is part of the interagency Multi-Resolution Land Characterization initiative to provide a nationally consistent land cover data set for the USGS, the Environmental Protection Agency, the National Oceanic and

Atmospheric Administration, and the U.S. Forest Service. NLCD classifies every location in the U.S. as one of 21 types of land cover classes (LCC) based on Landsat 7 Thematic Mapper imagery, topographic, census, agricultural, soil, and wetland data, and other land cover maps. In an orthorectified NLCD GeoTIFF image (as shown in Figure 2-4), each pixel represents a discrete portion of the Earth (30m resolution) with a single classification value. The USGS's Seamless Data Distribution System offers users the ability to freely download NLCD images from the Internet.

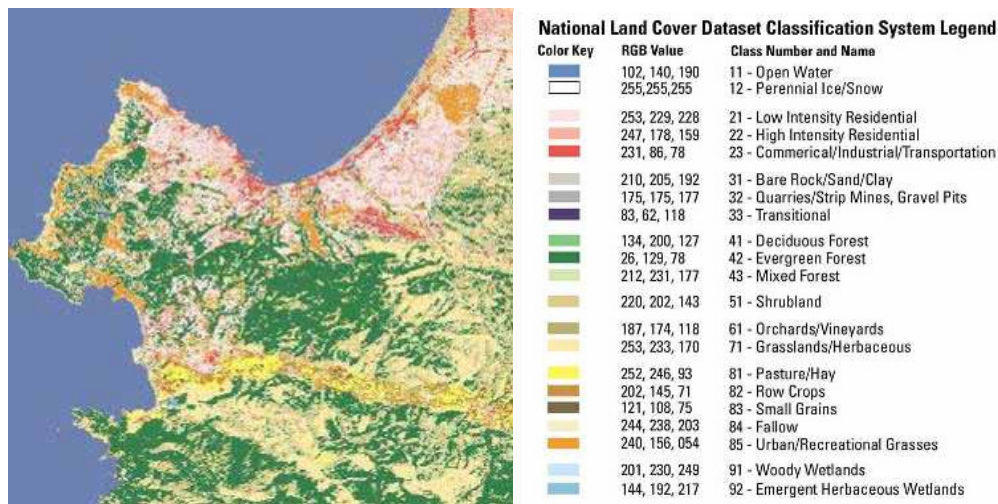


Figure 2-4 NLCD map of Monterey, CA with legend

E. VEGETATION PLACEMENT WITHIN COMMERCIAL TOOLS

Terrain creation tools automate some of the tasks associated with reading in these datasets and placing terrain objects within the environment. Users typically choose between the generation of polygonal forest objects, random placement of vegetation objects within a vegetation region, or manual placement of individual vegetation objects. The first method creates vegetation “blocks” (see Figure 2-5). The sharp angles, lack of transition zones, repetition of textures and inability to move within these forest mesas (passing through reveals the emptiness within) destroys believability at low to medium altitudes. Additionally, tree blocks must be “cut” to allow road and communication networks to pass through (see Figure 2-6). Random placement of vegetation objects within a polygonal area often results in sparse dispersions of identical objects. These

coarse representations are a result of most datasets lacking the ability to represent multiple overlapping land cover types within the same region. Vegetation sparseness arises from a combination of performance constraints and an inability to show natural patterns within the actual ecosystem due to poor dataset resolution. Manual placement of millions of vegetation objects is prohibitively time-consuming to consider for large-scale environments. Manual placement and recent automated feature extraction (with manual manipulation) is currently the only way to guarantee geospecific vegetation placement.



Figure 2-5 Example of a vegetation block

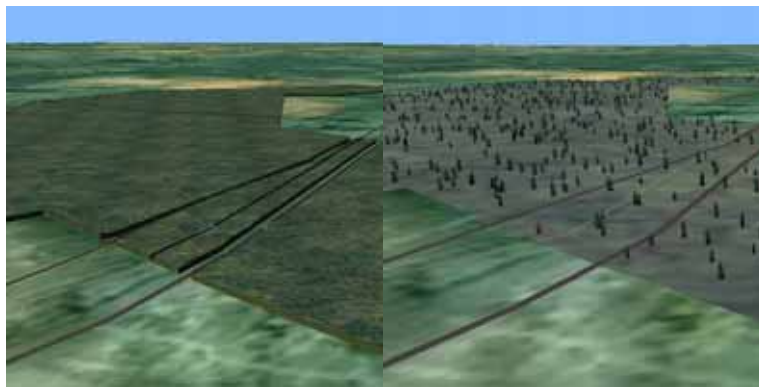


Figure 2-6 Comparison of vegetation block vs. sparse distribution of objects

F. MODELING PLANT DISTRIBUTIONS

As opposed to the random Poisson (a.k.a. dart throwing) distributions usually seen in commercial applications, academic solutions to creating large-scale plant distributions have ranged from detailed feature extraction, to geotypical ecotype modeling, to adaptive statistical placement, to plant growth simulation, to plant instancing using aperiodic tiling techniques. We will look at each of these methods in this section.

1. Semi-Automated Landscape Feature Extraction

Georgia Tech researchers developed a semi-automated procedure for correctly placing 3D tree objects using overhead imagery. (Wasilewski et al., 2002) Their technique requires the subdivision of large, very high resolution (e.g. 6 inch) geocorrected and geolocated imagery into manageable sub-images (512x512 pixels; 256x256 feet). A user manually selects tree areas as training data from one or more of these sub-images. Color statistics of these sampled areas are computed and similar tree areas are detected in subsequent images. A special thinning algorithm helps to define tree group blobs which are narrowed to lines to retain the topology of the blob and determine individual tree locations within tree groups. Magnitudes of the blobs are used to scale the radii of the tree objects. Grossly overlapping trees are culled based on a comparison of tree-tree distance to combined radii. Tree color is randomly selected based on the distribution of sample tree pixels, and height is estimated from tree radius. The final tree objects (i.e. cross polygon billboards) are then inserted into the terrain database.

This solution, while useful for generating vegetation within urban environments, would not work for large-scale rural vegetation placement and densely vegetated terrains. A heavily forested area may be devoid of the necessary clearings required by the thinning algorithm to determine the shape of individual trees. In such a situation, the placement of tree objects would be randomized. As seen in Figure 2-7, while their algorithm is able to recreate the basic shape of the upper center grouping of trees, the exact placement of tree objects is questionable (e.g. note numerous trees placed in the middle of roads). In the bottom left of the image, tree shadows and green grass have conspired to fool the algorithm into incorrectly placing trees. The rest of the image shows the usefulness of this technique: to generate tree-lined streets and placement of individual trees in an urban environment.



Figure 2-7 Geospecific tree placement using feature extraction and modeling
(From Wasilewski et al.)

The final concern about this technique is its applicability and scalability. Sub-meter imagery is generally limited to major urban centers within North America, Europe and those areas of recent military interest in Southwest Asia. 6-inch imagery of rural areas is nearly impossible to obtain and of questionable value as described previously (e.g. seasonal and temporal factors, storage requirements, etc.). The standard 1 degree by 1 degree geocell (nominally 100 km x 100 km) used in large scale terrain visualization would require the processing of over 1.6 million 512x512 images using 6-inch imagery.

2. Ecotope Modeling

Johan Hammes developed a real-time approach to generating plant distributions by using elevation-derived topographic information (i.e. elevation, slope, aspect, relative elevation) and multifractal noise to determine localized ecotopes (see Table 2-1). (Hammes, 2001) An ecotope is a particular habitat within a region with relatively uniform climatological and soil conditions. Typically, specific ecotopes will be associated with specific ecosystems. This includes all the plants and the associated

ground cover textures. As the user moves around the environment, the ecotope simulation is done for each new area that appears on screen and an appropriate ground cover and set of plants is generated.

Table 2-1 Variables used to calculate vegetation placement (From Hammes)

Elevation	The height above sea level. With increases in elevation, the general conditions become harsher. All plants have an upper limit at which they can survive. Plants also tend to become smaller with increases in altitude.
Relative elevation	Relative elevation refers to the local changes in height, with negative values showing depressions, valleys etc, and positive values showing ridges. This is the higher frequencies of the terrain. Relative altitude affects plant growth since valleys are generally wetter, as well as more sheltered. Ridges on the other hand tend to be exposed to the elements much more.
Slope	The slope of the terrain has a direct bearing on the quality and depth of the soil, as well as water retention due to runoff. Steep slopes tend to have small shrubs and grass cover. Very steep slopes tend to be exposed rock with no vegetation.
Slope direction (a.k.a. aspect angle)	The direction that the slope faces has a direct bearing on how many sunlight hours it receives, as well as being more sheltered or exposed to the prevailing winds.
Multi-fractal noise	Some plants and ecosystems also exhibit local grouping behavior independent of the above 4 variables. One reason is reproductive behavior. Plants that either drop their seeds, or reproduce vegetatively from roots tend to exhibit strong grouping behavior. A lot of multi-fractal noise functions exhibit similar patterns, and can be used to change the probability of ecosystems, or the density distribution of plants within ecosystems, to model this behavior.

The advantages of the Hammes approach were that the algorithms were sufficiently fast enough on modern personal computers to allow for real-time computation while maintaining interactive frame rates and that the resulting placement of vegetation looked natural while remaining deterministic (see Figure 2-8). One problem with his algorithm is that it did not take real-world land cover data into account; resulting in plausible, though generic, geotypical terrains. Additionally, each layer (from big trees to small rocks and plants) of the multilayer-based ecosystem's had its own defined set of characteristics (e.g. topographic regimes, list of possible vegetation objects, ground cover textures, object density, plant size, tile size). Populating this database without intimate knowledge of the terrain would be problematic. Finally, the scalability of the system to calculate ecotopes in real-time for large-scale terrains was never tested. The small-scale test conducted used single-layer ecosystems and only a single object from one ecosystem was placed within the terrain.

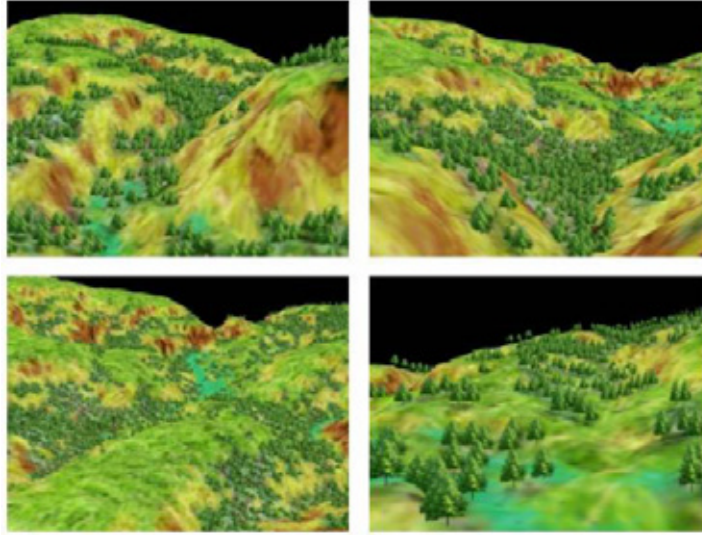


Figure 2-8 Example of ecotope modeling
(From Hammes)

3. Adaptive Statistical Techniques

A modification of the random Poisson distribution used within many terrain database tools is the Poisson disk distribution whereby each vegetation location point p must be further than some radial distance r away from all the other points. The system checks on each potential point and rejects those that do not meet the above criteria. This procedure works well for distributions with low point densities, but as the number of existing points grows, the placement of satisfactory points becomes increasingly more difficult; slowing down the efficiency of the algorithm and eventually leading to the case where another valid point cannot be placed.

In order to optimize the number of points that can be introduced to an area and still satisfy the Poisson disk criteria, one can turn to using Voronoi regions. (Okabe, Boots, & Sugihara, 1992) In this case, each point p_i is assigned a localized area that contains all the points in the plane that are closer to the given point p_i than any other point in the existing point set. A spiderweb-like mesh is created with the lines between vertices representing the equidistant boundaries of neighboring points (see Figure 2-9). This condition in itself does not satisfy the Poisson disk distribution criteria since we do not restrict the placement of new points. However, if each point was moved to the centroid

of its enclosed Voronoi region in an iterative fashion, known as Lloyd's method (Okabe et al., 1992), one would get a resulting distribution that is evenly spread (i.e. similarly sized Voronoi regions) throughout the entire area (see Figure 2-10).

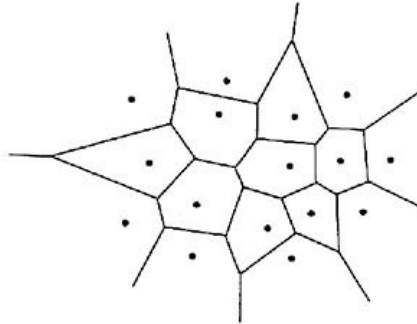


Figure 2-9 Example of Voronoi regions

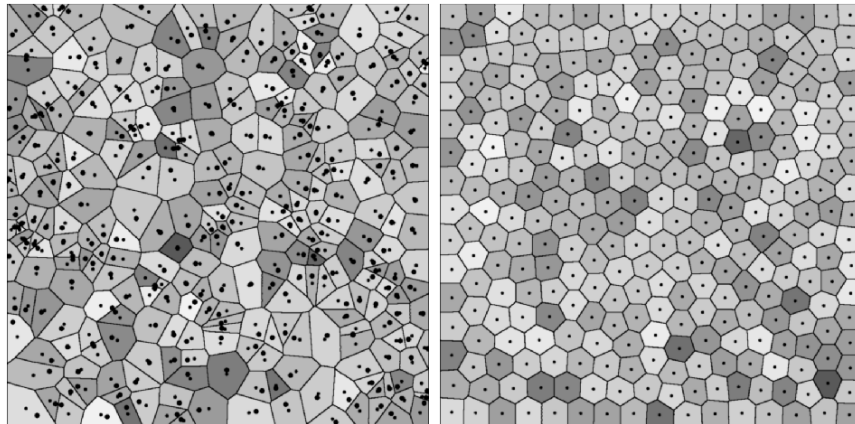


Figure 2-10 Voronoi regions before movement of points (big dots) to centroids (small dots) and after using Lloyd's method

The even spacing of this technique limits its use to areas where the localized density (i.e. “clumping”) of individual objects is not required such as grassy fields or dense woods. A way to bring back density variations and vegetation variety is to use the calculated positions for the placement of vegetation groupings of multiple vegetation objects. In Figure 2-10, if each of the different shades of gray represented a different grouping or tile of vegetation objects, we could overcome the uniform nature of the resulting distribution while filling the entire region with vegetation objects.

4. Plant Growth Simulation

The simulation of plant populations requires the modeling of entities competing for resources (e.g. light, water, nutrition) and growth rate as determined as a function of population size. Given some basic limits to population growth (e.g. a fixed area), the plant population will grow until an equilibrium point is reached, at which point some plants will start to die off. This self-thinning process has been examined in detail by Prusinkiewicz. (Prusinkiewicz, 1999, 2000; Prusinkiewicz, Mündermann, Karwowski, & Lane, 2001) In one case, circles (akin to the bounding volumes of geometric objects used within GENETICS) were randomly distributed across a plane with random diameters (see Figure 2-11). Growth took place as successive increases in circle size. If two circles collided, survival of the fittest occurred and the smaller circle “died”. This technique was also extended to multiple plant species using several competitive resources, plant attributes, and probabilities of death as seen in Figure 2-12. It can also be used for determining survival criteria for vegetation object placement within GENETICS.

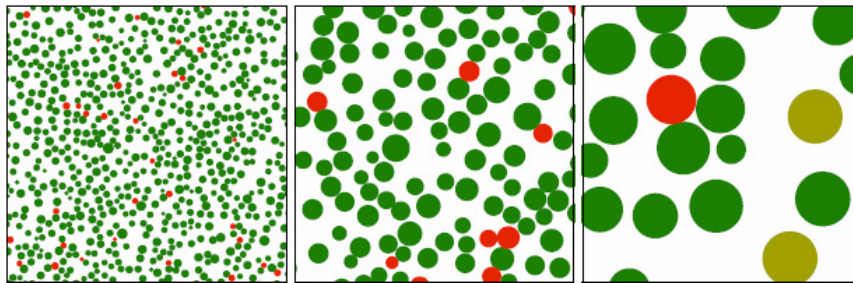


Figure 2-11 Simulated growth of a self-thinning plant population; red circles die, gold circles are dormant
(From Prusinkiewicz et al.)

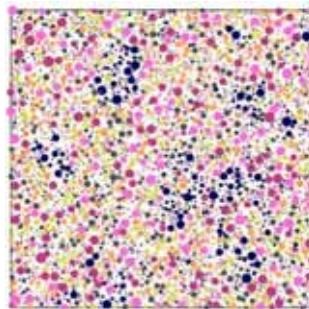


Figure 2-12 Simulated plant population of eight different species
(From Prusinkiewicz et al.)

5. Aperiodic Tiling

Wang Tiles, named after Hao Wang, consist of a set of squares in which each edge of each tile is assigned a color. (Wang, 1961, 1965) These squares cannot be rotated. A valid tiling requires all shared edges between tiles to have matching colors. As few as eight tiles can be used with a simple stochastic system to create a non-periodic tiling plane (see Figure 2-13). The tiles may be filled with 2D textures (and Poisson distributions with 3D geometry as shown soon) that when assembled create a continuous representation. The main advantage of using Wang Tiles is that once the tiles are filled, large expanses of an aperiodic texture (or distributions of geometry) can be created as needed very efficiently at runtime. Such a technique is useful for creating large arrangements of plants or other objects on a terrain.

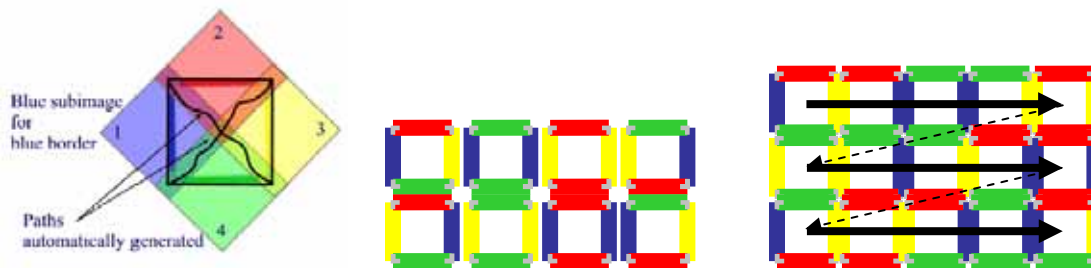


Figure 2-13 Wang Tiles: construction, minimum non-periodic tile set and Wang tiling texture generation
(From Cohen et al.)

To create a Wang Tile, textures must be found that fit together across the boundaries with matching colors. One way to create such a tile is by combining random diamond shaped (i.e. squares rotated by 45 degrees) sample portions of the source image, one for each edge color of horizontal and vertical edges. For the minimum set of eight tiles, we will need four sample images. For a set of 18 tiles (expanded to reduce repetition artifacts), we will need six images. The challenge is to create four cutting paths between the overlapping tiles. If a reasonable cutting path cannot be obtained (determined by a pixel color error metric over the set of 32 paths for the eight tile set), then a new set of four sample diamonds is selected.

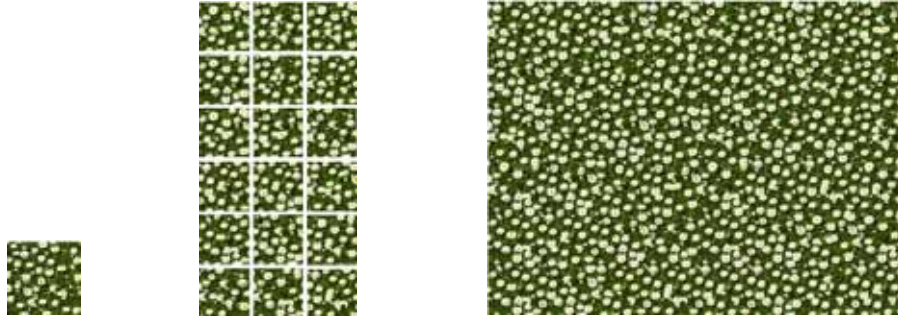


Figure 2-14 Example of Wang Tiling: input texture, automatically generated set of tiles, & resultant synthesized texture
(From Cohen et al.)

Cohen, Shade, Hiller, and Deussen extended the concept of Wang Tiles by including corners into the tiling algorithm to cover features that cross beyond a single edge. (Cohen, Shade, Hiller, & Deussen, 2003) This additional constraint which added the need for vastly more tiles (two colors per corner raised to the power of four corners = 16 corner coding possibilities per tile, times a minimum set of eight tiles = 128 corner-coded tiles) also added the freedom of modulating the texture by mixing two source textures. This allows for inhomogeneous tiling (e.g. a high density flower/low density grass texture mixed with a low density flower/high density grass texture results in a Wang Tile plane of flowers and grass (see Figure 2-14). Cohen et. al. take the concept of Wang Tiles a step farther by using them to create Poisson-like pseudo random distributions of 3D objects. Instead of a typical texture image, a Poisson disk of points serves as the basis of the texture. From this source texture, a much larger arrangement of evenly distributed points can be created. These points can act as the position markers for 3D objects (e.g. flowers in a field). The description for how to accomplish this task as presented in their paper suffers from several faults. First, multiple variations (i.e. size, shape, orientation, etc.) of the same type of object must be stochastically placed within each of the eight Wang Tiles based on the Poisson distribution concept. The second flaw is the need for an elaborate view-dependent level of detailing scheme using a hierarchy of layered depth image textures. Thus, considerable time must be spent in offline rendering of the object from a variety of lighting conditions, viewpoints, and distances/resolutions.

Cohen et. al. were able to render a large daisy scene at 3-4 fps using a software renderer and estimated that hardware rendering could achieve 30 fps.

G. NATURAL ENVIRONMENT RENDERING

An excellent survey of natural environment rendering algorithms is presented in “The State of the Art in Realtime Rendering of Vegetation” by Mantler, Tobler, and Fuhrmann published in July 2003. (Mantler, Tobler, & Fuhrmann, 2003) Several of the techniques presented here were touched on in their paper. Since billboarding and imposters form the basis for many of these methods, they are given special attention in the description that follows.

1. Billboards & Imposters

The concept behind the use of billboards is to minimize the number of polygons and allow the texture map to represent all the details of the object in question. Typically, this is either done with a single screen-aligned or axial-aligned flat polygon (polygon always faces in the direction of the user or is bound to the terrain surface with a fixed world up vector) or a cross-polygon scheme where two or more polygons intersect along the vertical axis of the object in question. In the former method, the polygon is always turned towards the viewer while in the later they are not. Billboarded images generally make use of an alpha channel to represent the silhouette of the rendered image. While it is difficult to detect the use of billboards for distant objects, closer inspection will reveal the trick. Thus, it is best to use geometric models for nearby objects and use billboards for background objects. Of course, this doesn’t solve the problem of flying over a billboarded forest. While the undesired cutout effect can be somewhat alleviated by using multiple cross-polygon axially-aligned billboarded trees and a circular horizontal cross-section texture, the effect is still there and now creates the need to have symmetric trees. A way around this problem is by using imposters. (Forsyth, 2001)

Imposters are similar to billboards and use either prerendered textures or ones created at runtime. Prerendered textures, popular in the gaming community, occur off-line and consist of object renderings from a range of angles to reconstruct the object’s appearance. Using a minimum number of polygons and these texture maps we can

represent geometry from each particular viewing angle. For distant renderings of upright objects, the number of angles around the vertical axis can range from as few as four up to sixteen or more. At render time, the application determines the angle between the viewer and the object's forward axis and selects the proper texture frame to apply to the imposter. For small changes in viewing angle, the imposter image acts like a billboard with a locked vertical axis. As the viewer travels around the object, different angled versions of the rendered object are used as the billboard's texture. Blending between old and new textures can reduce the effect of texture popping. If the viewer gets into a position to view the object from an angle significantly higher or lower, then switching to a geometric representation of the object is an option or a cloud of imposters can be used (discussed later). Of course it is also possible to generate imposters during runtime, depending on the complexity and number of imposters that must be created. In this case, the object in question is rendered to an empty texture object where the background is transparent and then applied to a screen-aligned billboard. The imposter is usually valid for a number of frames before it must be updated and the process starts again. Care must be taken to ensure the texture resolution is matched to the approximate screen size of the impostered object within the scene. A small texture applied to a larger sized object will look pixellated while a detailed rendering will be wasted on a tiny object.

One of the problems with both billboards and imposters is object animation. The multiplicative cost of rendering the image from a range of angles and frames of animation becomes quickly prohibitive. Thus, animation is typically limited to a three or four frame sequence. Effectively this turns object animation into two states: "moving" or "not moving." This does not prevent a billboard or imposter from taking part in a larger animation. Such is the case for a group of leaves and small branches, represented by a billboard, being moved by a geometric model of the tree's trunk and branches.

One of the more interesting geometry-reducing techniques presented recently is found within the paper "Billboard Clouds for Extreme Model Simplification" by Decoret, Durand, Sillion & Dorsey. (Decoret, Durand, Sillion, & Dorsey, 2003) The concept is that a complex model can be simplified into a set of independent (in size, orientation, and texture resolution) planes with texture and transparency maps. Decoret et. al. show an

optimization approach to build the billboard cloud given a geometric error threshold. A greedy algorithm selects suitable representative planes based on a density function that maximizes the number of faces that can be projected onto a particular plane. Planes that are nearly tangent to large faces on the model are favored to create a good surface approximation. Cracks are avoided by projecting primitives and their respective textures onto multiple planes.

The billboard cloud results presented at SIGGRAPH 2003 showed that complex objects of thousands of polygons could be simplified to approximately 100 planes (using a 6% error bound). A problem with creating billboard clouds is that since the algorithm works in $O(kn)$ time (where n is the size of the input mesh and k is the number of planes in the billboard cloud), it is unsuitable to make such calculations during a real-time constrained simulation loop, thus billboard clouds must remain relatively static since they can only be created during preprocessing. The authors also propose a way to further simplify complex objects by creating view-dependent billboard clouds. This would create an imposter cloud valid only over a particular range of viewing angles and viewing distances. As the viewer passes beyond this range, a new view-dependent imposter cloud must be created (keeping in mind real-time simulation constraints as before) or loaded into memory if preprocessed (see Figure 2-15).



Figure 2-15 Example of view-dependent billboards. Billboard clouds are built for each view-cell. Object distance determines texture resolution & number of billboards needed to represent the shape and appearance of original model.

(From Decoret et al.)

2. Point/Line Clouds

Since 1985, points and lines have been recognized as a viable and efficient way to render vegetation. Of course, in 1985, it took 5-10 hours on a VAX 11/750 to create an image of a forest. (Mantler et al., 2003) Today, research into simulating vegetation has largely come from synthetic plant expert Oliver Deussen whose impressive list of collaborations include (Cohen et al., 2003; Deussen, Colditz, Stamminger, & Drettakis, 2002; Deussen et al., 1998; Deussen & Lintermann, 1997; Lintermann & Deussen, 1996a, 1996b, 1997, 1999). His paper on using point and line clouds for distant vegetation, “Interactive Representation of Complex Ecosystems,” describes building matching display lists for a tree object; one for polygonal data, one for point/line data. (Deussen et al., 2002) Random (but synchronized) reordering of these lists allows for the renderer to partially render the polygonal display list and then switch to the point/line list, while guaranteeing that the entire model will be covered. Switching determination is based on whether a point representation can match the polygonal representation with no holes and correct coverage. Thus, the system avoids localized popping on the tree by randomly distributing the transitions of a tree’s geometry to lines and points.

3. Multiresolution Modeling

Remolar et. al. present a multiresolution technique of rendering Xfrog-modeled trees in real-time using dynamically generated imposters. (I. Remolar, M. Chover, O. Belmonte, J. Ribelles, & C. Rebollo, 2002; I. Remolar, M. Chover, Ó. Belmonte, J. Ribelles, & C. Rebollo, 2002; Ribelles, 2003) Frame-to-frame coherence means that imposters can be used over the course of several frames (so long as the image stays within an tolerance error threshold). The continuous multiresolution modeling approach allows for less precise rendering of trees far away, and for trees closer to the viewer, less precise rendering of the bulk of the leaves that form the core of the tree and high precision rendering of leaves on the outer edge. Imposters can also be combined with geometry (i.e. front of tree, geometry; back of tree, imposter), producing a 3D effect missing from most image-based rendering. Trees are separated into two components: solid mass (trunk and branches) and foliage or leaves. The solid pieces are represented with a polygonal mesh that can be reduced in detail. Each leaf is represented by a

textured quadrilateral and the collective grouping of these independent polygons are simplified through a method based on “View Dependent Refinement of Progressive Meshes” by Hoppe (Hoppe, 1997): the Leaf Simplification Algorithm (LSA). With LSA, two leaves are replaced with a new one that preserves an area similar to that of the collapsed leaves. Also, in a preprocessing step, leaf collapse sequences generate binary trees with the root nodes being the minimum polygonal representation able to retain the shape of the foliage and the individual leaves being the leaf nodes of the trees. Using the multiresolution modeling combined with imposters, Remolar et. al. were able to generate over 200 detailed trees at 15 fps. No effort was made to realistically light the trees or include shadowing.

H. TERRAIN MESH GENERATION

While our research principally concerns the visualization of ecosystems, it will be necessary to place an ecotope’s vegetation objects on top of a terrain surface mesh. While these meshes generally represent a visualization of an existing dataset (e.g. a grid of measured elevation postings), we will be interested in filling in data between known elevation postings to generate the appearance of higher fidelity data or fill in missing postings. The following are well known techniques for generating plausible terrain surface data.

1. Heightmaps

To start our discussion of terrain mesh generation, we should first take time to talk about the creation and use of heightmaps. Heightmaps are created at run-time from numeric datasets or loaded from a grayscale image where the number of possible shades of gray (typically [0-255]) is equivalent to the number of height levels in the terrain. (Polack, 2002) Where more precision is required, an RGB encoding scheme is used to increase the number of height levels to over 16 million. Since both computer images and elevation postings fall in a regular grid pattern, it is easy to take the color value of a pixel at a particular (S, T) image location and convert it into an elevation value (Z) at a corresponding world location (X, Y). The “distance” between pixel-encoded elevation

values represents the spacing between the vertices in our world-space grid. Ideally, for real-world elevation grids, each measured data point is presented by its own pixel.

The simplest terrain rendering algorithms will load a heightmap image, create a 3D mesh based on the heightmap, and render the terrain using all vertices generated. We will talk more about heightmaps in the next section on terrain rendering and level of detail. But first, we need to address the creation of plausible terrain heightmaps using fractal terrain generation.

2. Fractal Terrain Generation

a. Fault Formation

“Fault formation” is a fractal terrain generation technique used to simulate plate tectonics and erosion. (Krtén, 1994; Polack, 2002; Shankel, 2000a) By generating “faults” in the terrain and applying an erosion filter to soften the breaks, a surprisingly realistic, albeit fairly “smooth” heightmap can be achieved. Note that there is no dimension restriction (i.e. terrains need not be square or a power of two).

One fault formation technique works by choosing two points at random and drawing a line between them. One randomly selected side is given an increase in height (i.e. lighter shade of gray). Next, two more points are chosen and again a line is drawn between them. This time, the random side that gets the height increase receives a smaller increase. This is repeated until there are a predetermined number of lines on the screen (decreasing the added height with each iteration). The next step is to apply a subdivision technique (e.g. a low-pass image filter) to smooth the high frequency differences between the angular regions of gray. To get a more jagged terrain, eliminate the “reduce height each iteration” requirement or simply reduce the smoothing/blurring ability of the erosion filter.

b. Midpoint Displacement

Fault formation is useful for creating smooth, rolling hills, but if the scene requires a more chaotic pattern like a mountain range, then midpoint displacement is a fractal terrain generation technique that could work. The idea behind midpoint displacement is to simulate the uplift in the terrain. (Polack, 2002; Shankel, 2000b)

Unlike fault formation, midpoint displacement will create a square, power of two heightmap (e.g. 512x512).

One midpoint displacement method starts with a empty square terrain space. The midpoints of each edge are found to in order to find the center of the square. This center point is given a height value by averaging the corner heights and adding a random height within the range of $-fHeight/2$ to $fHeight/2$ where $fHeight$ is defined as either the maximum number of color values (256 for a grayscale heightmap) or the length of the longest edge. Edge midpoint heights are determined by averaging the heights of the nearest neighbors (corners on the edge and the center point) and randomly adding a height value as before. This process is repeated for the four new squares. In order to control the variability of the terrain, the $fHeight$ value is reduced each iteration. This reduction can be performed by multiplying the current $fHeight$ value by $2^{-fRoughness}$ where $fRoughness$ is a parameter controlling the smoothness of the terrain (values under 1.0 are increasingly chaotic, values above 1.0 are increasingly smooth).

c. Particle Deposition

In nature, volcanic mountain ranges and island systems are generated by lava flow. We can simulate this effect by using a particle system to drop sequences of particles and simulate their flow across a surface composed of previously-dropped particles.(Shankel, 2000c) Dropping a sufficient number of particles will produce structures that look like the flow patterns of a viscous fluid (lava).

The algorithm is very straightforward. We drop a single particle representing an incremental increase in height (i.e. brightness) on an empty height field (i.e. all black). A second particle is dropped on the first and perturbed until it comes to rest. This ensures that none of its neighbors is at a lower altitude. Particles continue to drop and the drop point is varied until a significant pile is created. The terrain designer controls the terrain's shape through movement of the drop point. A large peak forms by keeping the drop point in a single location. Chains of multiple peaks are created by moving the drop point periodically. The terrain designer can create volcano-like structures by designating a caldera line. This is a height level on a peak where the upper portion of the peak is symmetrically inverted to create a crater. Further particles within

this region can have the effect of deepening the crater or can (after a certain depth is reached) be used to create a cinder cone within the caldera. An erosion filter similar to the one used with the fault formation technique can be employed to soften any hard edges.

d. Fractal Brownian Motion

Fractal Brownian Motion (fBm) is a fractal sum of a noise function of the form: $noise(p) + \frac{1}{2} * noise(2*p) + \frac{1}{4} * noise(4*p) + \dots$ (Ebert, 2003; Musgrave, 1993; O'Neil, 2001) While any noise function can be used in calculating the fractal sum, Perlin noise (Perlin, 1985) is probably the most commonly used fBm noise function since it is a fast way of generating high-quality noise. Perlin noise is considered a key ingredient for creating procedural textures and landscapes. It uses a pseudo-random number generator (PRNG) to create a noise texture or look-up table, and there are ways of manipulating the noise function to make its output look interrelated yet natural and random. The pseudo-random feature of Perlin noise is that the function outputs the same number every time given the same input. (Lecky-Thompson, 1999, 2000)

An fBm function is typically implemented as a loop and the number of times through the loop is called the octave. Note that using the same input parameters to the noise function will result in the same fBm output and with each increase in fBm octave, the texture pattern is perturbed to a finer level of detail. As with Perlin noise, we can zoom in or out of any part of the texture by changing the range of numbers passed to the noise function. The closer we zoom in, the higher the number of octaves needed to maintain a good level of detail and complexity in the image. We can also manipulate fBm output by raising the noise function output to a fractional exponent or by changing the lacunarity factor (how parameters are scaled with each octave). Exponents act as a roughness factor and range from 0.0 (very rough) to 1.0 (very smooth). Lacunarity values between 1.0 and 2.0 provide a recursive feedback (making for more detailed noise at performance cost), values below 1.0 reduce the noise range with each octave (coarser noise added with more weight to the image), and values higher than 2.0 cause the noise range to increase more quickly (finer noise added with more weight to the image). Ken Musgrave, founder of MojoWorlds (a virtual terrain visualization software package), recommends values of 1.9-2.2 for terrain generation.

e. Multifractals

Multifractals are basically just a more complex form of fBm. (O'Neil, 2001) Some multifractals use a fractal product instead of a fractal sum (i.e. multiplying instead of adding noise components together). Some add variable offsets or apply other mathematical functions within the loop (e.g. `abs()`, `pow()`, `exp()` or trigonometric function). Multifractals are useful within terrain visualization since certain terrain features can be locally manipulated for a more realistic effect.(Musgrave, 1993) For example, land close to sea level can be flattened and smoothed while land at higher altitudes can be made more mountainous and chaotic. For more examples of multifractals, see Ken Musgrave's chapter within *Texturing & Modeling: A Procedural Approach*. (Ebert, 2003)

I. TERRAIN MESH RENDERING & LEVEL OF DETAIL

Terrain rendering is not the easiest of subjects to discuss. In order to bring the reader up to speed on terminology and issues associated with LOD techniques, a few topics must be covered to include the geometric representation of terrain surfaces, a short discussion on image-based representations of terrain surfaces, and an introduction into terrain level-of-detailing that will include sections on error metrics, two basic types of LOD techniques, view-dependent refinements, geomorphing, occlusion culling, and fixing cracks. All of this is necessary before a discussion of the pros and cons of various LOD schemes can be presented.

1. Geometric Representation of the Surface

Before we can survey how terrains are actually rendered to the screen, we need to look at how terrain data is represented. From a geometric point of view, a terrain can be defined as a generic three dimensional surface using a bivariate vector mapping function $p: R^2 \rightarrow R^3$ like $p(i, j) = [x(i, j), y(i, j), z(i, j)]$ where two parameters (i and j) drive each component of the p vector.(Balogh, 2003) These functions are generally not analytically defined, but rather their domain and range are quantized into regular discrete intervals and the functions are given numerically as one large multidimensional array. This means that our elevation data will consist of a given set of points instead of a truly continuous

surface. In order to preserve the surface's C^0 continuity (i.e. surface going through the specified points), we need to utilize an interpolation method. One of the simplest forms of interpolation is the linear one, that of patching the hole between three neighboring points with a triangle. While simplistic (versus using Bezier curves and other non-linear interpolation techniques), this method works well for today's hardware-assisted rendering and is complementary to the midpoint displacement algorithm presented earlier. Additionally, one triangle can be tessellated into many smaller triangles, thus enabling a way to simulate nonlinear interpolation methods through vertex displacement.

So far we have defined the function's domain as being two dimensional. This allows us to interpret the data as points defined over an imaginary grid characterized by the parameters i and j . This is important because a grid gives us the connectivity information required to correctly interpolate between discrete points; unlike Triangular Irregular Networks (TINs) that cannot be given by the function above and require special connectivity information.(Luebke, 2003)

2. Heightmaps Revisited

In the previous section, we briefly discussed the use and generation of heightmaps. Now we will examine heightmaps within the context of the vector function given above and their relationship to displacement maps.

While the vector function described previously gives us the most flexibility when defining a surface, in most cases the terrain is not given as such a generic surface but as simple elevation data called a heightmap. This can be described with a simpler mapping function where $p(x,y) = [x, y, z(x,y)]$. Clearly, this is just a constrained version of the previous surface equation since overhanging features (see Figure 2-16) cannot be represented. Even with this constraint, the form is preferred for its efficiency in memory usage since we only need to store a single scalar value for each point in the grid. The other two values are implicitly stored as indices to the array.

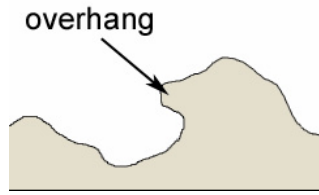


Figure 2-16 Overhanging terrain
(From Balogh)

Generalizing this idea means that we can use a simple vector function to describe the base form of the surface and another compact scalar function to change that base surface. An example of adding detail to a base surface in this manner is called displacement mapping. Displacement mapping shifts each point on a base surface in the direction of a normal vector n by an amount given by a bivariate scalar mapping function $d: R^2 \rightarrow R$. Thus, the final displaced position is $r(i, j) = p(i, j) + n(i, j)d(i, j)$. The perceptive reader will recognize that a heightmap is really just a displacement map defined over a flat plane.

3. Level of Detail

It should be intuitively obvious (“...to the most casual observer”) that rendering the massive number of the triangles generated by our heightmap-encoded datapoints or one of our fractal terrain generation algorithms is terribly inefficient. Thus, it is necessary to find a way to display these huge terrain heightfields by accessing and using just a limited number of vertices. This inevitably means that we have to reduce detail (i.e. vertices and triangles). The ultimate goal is to drop as much unnecessary detail as possible while still preserving a good level of perceived image quality. In general, these kinds of algorithms are collectively called Level of Detail (LOD) algorithms.

Figure 2-17 demonstrates the problem caused by the lack of LOD. Every triangle on the screen has approximately the same size in world space. The foreground (which makes up the majority of the pixels on the screen) is composed of only a few triangles lacks any notion of detail. A little farther back, there is a dramatic density change, where the terrain surface is almost parallel to the viewing plane (i.e. the surface normal is perpendicular to the viewing vector), thus triangles located there are very small when

projected to screen space and have a minimal contribution to the final image. Still farther back, projected triangles are also very small as a result of perspective. Clearly, we need a procedure in place to add detail to the foreground image while reducing detail in the background. But first we need to find an automatic way to determine which triangles should be tessellated and which should be merged or culled.

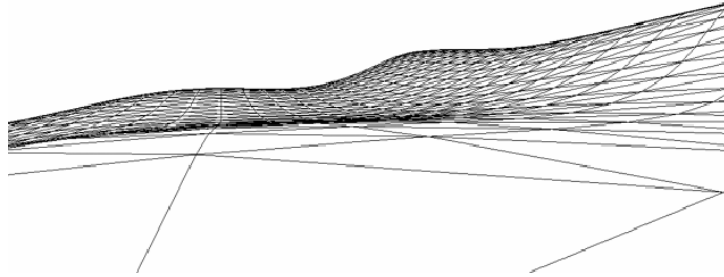


Figure 2-17 Example of terrain without LOD
(From Balogh)

a. Error Metrics

By using an error metric, we can measure how good our LOD approximation is for some part of the scene. Based on this error we can decide if we need to drop or add more detail. One method to measure error is to calculate a vertex's object space relative error (i.e. the difference vector between two consecutive LODs in object space). Since object space errors do not take perspective into account, we must project the object space error into screen space. The resulting screen space error effectively tells us how different the final image would be if we had included that vertex. Projections fall into two main types: anisotropic projections depend on viewing direction, while isotropic projections do not. Anisotropic projection yields more accurate results, but is usually more expensive to compute. Evaluating the error term for every vertex is not always practical and often it is better to calculate the errors for triangles or even whole meshes. In either case, the goal is to meet a given threshold of error using as few triangles as possible.

b. Discrete vs. Continuous Level of Detail (LOD)

There are two major kinds of LOD schemes: discrete and continuous. Discrete LODs have a fixed number of static meshes with different resolutions. During rendering, an appropriate mesh is selected based on its error metrics. The main advantage of this technique is that these static meshes can be precomputed up-front, allowing for mesh optimization without a performance hit to the CPU. Discrete LOD is fast, simple, and works well for meshes that are relatively far away from the viewer since the distance makes mesh selection practically view-independent. It does not work well for close-up viewpoints that require different tessellation levels at different parts of the mesh, depending on the view. Supporting view-dependent meshes with discrete LOD would require storing so many different static meshes that it would make this method impractical. Since terrain rendering requires close-up views of huge datasets, discrete LODs would seem to be a poor choice. However, it is possible to partition an object into smaller parts with view-independent LODs and render those separately. This technique has a couple of issues though: joining neighboring parts seamlessly is problematic and finding the right balance between the number of LOD meshes and mesh resolution is not easy and is highly application-specific. Although discrete LOD schemes are quite rigid, we will look at an example of an efficient terrain rendering method that uses discrete LOD.

In contrast, continuous LOD (CLOD) schemes offer a practically unlimited number of different meshes without requiring additional storage space for each of them. These algorithms assemble the representative mesh at runtime, giving the application much finer control over mesh approximation (i.e. require less triangles to achieve the same image quality). The downside is that CLOD algorithms are usually quite complex and the performance cost of building and maintaining the mesh is quite high. Additionally, since the mesh is dynamic, it must reside in system memory and be sent over to the GPU every frame. Even with these limitations, most terrain rendering algorithms are based on CLOD schemes because of its flexibility and scalability.

c. View-dependent Refinements

CLOD algorithms construct a triangle mesh that approximates the terrain as it would be seen from a particular location. This means that only visible vertices that play a significant role in forming the shape of the terrain should be selected into the mesh. Finer tessellation occurs in nearby terrain and characteristic terrain features. These kind of algorithms use view-dependent refinement methods. The result is a mesh that is more detailed in the areas of interest and less detailed everywhere else. Refinement methods can be grouped into two categories. The first one is bottom-up refinement (a.k.a. decimation), which starts from the highest resolution mesh and reduces its complexity until an error metric is met. The other one is called top-down refinement, which starts from a very low resolution base mesh and adds detail only where necessary. The bottom-up method gives more optimal triangulations (i.e. a better approximation given the same triangle budget), but the calculations required are proportional to the size of the input data. The top-down method, on the other hand, results in a slightly less optimal triangulation, but is insensitive to the size of the input data. Its calculation requirements depend only on the size of the output mesh, which is generally much lower than the input. Legacy algorithms were based on the bottom-up method because input data was small and rendering was expensive. (P. Lindstrom et al., 1996) Today, most algorithms are based on the top-down scheme, since the size of input datasets has increased by orders of magnitude and rendering performance is much higher. With current hardware, it is cheaper to send a few more triangles to the GPU than spend long CPU cycles deciding that some of those triangles were unnecessary.

d. Geomorphing

Every time a new vertex is added to or removed from a mesh, the triangles sharing that vertex will suddenly change (causing the infamous “popping” effect). (Akenine-Moller & Haines, 2002) Since the human brain is very sensitive to these changes and even a small amount of popping can be rather disturbing, it is important to reduce or eliminate these abrupt changes in vertex positions. Although we can tell if a vertex suddenly jumps, we cannot tell if a vertex is actually in its correct position or not. Geomorphing techniques are based on this fact and slowly move the new vertex to its

new position (undetectable to the viewer). There are two kinds of geomorphing methods: time and error driven. Error-based morphing is generally preferred, because it morphs only during camera movement and does not need to remember the state (e.g. time) for morphing vertices. Note that for per vertex lighting, not just the position but the normal vector also requires smooth morphing. The interpolation of the normal vector can result in the shifting of the triangle's color, which might be noticeable. It should be noted that geomorphing only makes sense when dealing with non-trivial error thresholds. If rendering a mesh with projected errors below one pixel, geomorphing is unnecessary.

e. Occlusion Culling

Occlusion culling is a well known technique for removing hidden details by searching for occluders within the scene (e.g. a nearby hill) and then culling away the details occluded by it. These methods perform well with scenes having considerable depth complexity (e.g. first-person shooters). The problem with this technique is that for the worst case, looking at the terrain from the vantage point of a high-flying aircraft, there will be minimal occlusion (i.e. all geometry having a depth complexity of one). Since we typically require our rendering algorithm to be fast enough to render at interactive frame rates in the worst case, there is seemingly little advantage in optimizing the best case (i.e. low to the ground). In the high-flier case, the occlusion culling algorithm will only add unnecessary overhead, although it can be used as a throttle for controlling the level of detail for other objects (e.g. vegetation) within the landscape. Thus, viewpoints closer to the ground would cull away more occluded terrain geometry, permitting more detailed objects on the terrain's surface.

f. Cracks

Cracks are an problem that must be solved with nearly all terrain CLOD algorithms and geomipmapping is no exception. Cracking happens along the shared edges where two different LODs meet and T-junctions have formed. Vertex layout and connectivity information typically must be rearranged dynamically get these shared edges to tightly fit together. Some algorithms (e.g. geomipmapping) avoid the costly adding or

deleting of vertices by using vertex indexing to simply change the vertices being processed and the resulting connectivity information.

Thatcher Ulrich spent a whole chapter discussing the various ways to fix cracks in his paper. (Ulrich, 2002) He identified three crack-fixing techniques: the ribbon, the flange, and the skirt. With ribbons, we insert a triangle into area to fix the crack. The obvious problem here is discovering the exact coordinates for the triangle's vertices. The next method, the flange, consists of creating and rotating a quad to cover the crack. Miscalculating the correct rotation can lead to the quad extending beyond the terrain or not filling the crack. Ulrich decided upon the skirt technique in his own LOD algorithm (e.g. Chunked LOD) which we'll describe later.

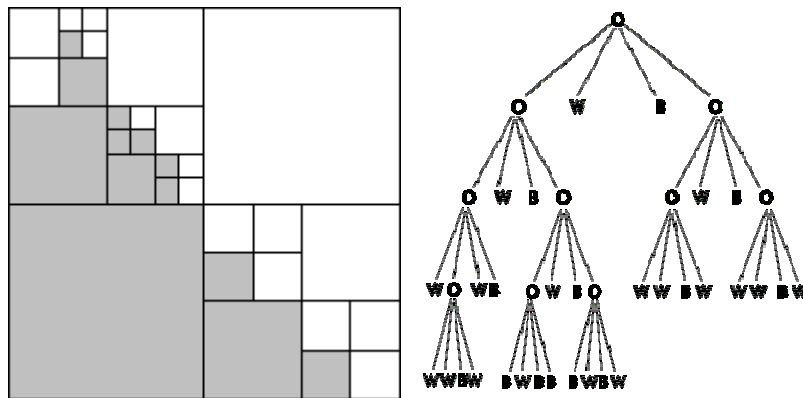


Figure 2-18 Quadtree representation of an image

4. Quadtrees

The LOD algorithms we will examine render terrains described by square grid-based heightfields. One straightforward method for partitioning a grid into smaller parts is by means of a quadtree (see Figure 2-18). Quadtrees are simply data structures, where each node has four children. When used to store terrain data, the root node represents the whole surface, and the child nodes subdivide the parent into four smaller partitions, and so on. Early terrain rendering techniques used this spatial partitioning to perform fast gross culling against the viewing frustum. (Pratt, 1993) While the algorithm has to subdivide down to the lowest level for rendering, large blocks of terrain can be quickly

culled away if completely outside the view frustum. It is easy to create an LOD scheme with this technique by subdividing deeper in branches with a higher error. This kind of subdivision tactic does not guarantee that the resulting mesh will be free cracks where different resolution parts connect. Restricted quadtree triangulation (RQT) solves this problem by walking a dependency graph defined over a grid and adding extra vertices when necessary. (Pajarola, 1998) Figure 2-19 illustrates this method.

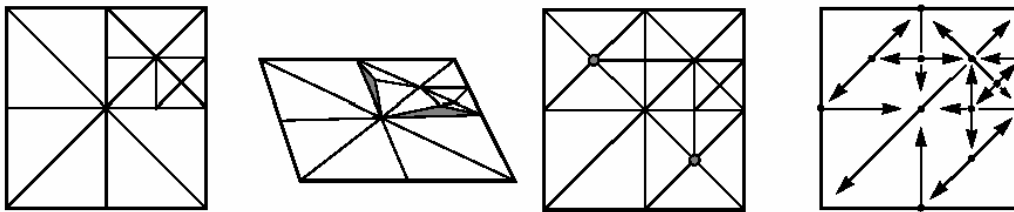


Figure 2-19 Walking the dependency graph and adding extra vertices where necessary results in a restricted quadtree triangulation
(From Pajarola)

There are a couple of issues with this technique, though. The explicit dependency walk is quite expensive, especially when dealing with a large number of vertices. Also, it only adds vertices, not triangles, and it is not trivial how to build a new triangulation for the resulting set of vertices.

5. Geomipmapping

Geomipmapping, developed by Willem H. de Boer, is a CLOD algorithm optimized for 3D graphics hardware that is similar to texture mipmapping except that terrain patches are substituted for texture patches.(de Boer, 2000) As an example, assume we have a 5x5 square terrain patch consisting of 25 vertices and 32 triangles. This mesh is labeled LOD 0. If we only use every other vertex along the edges of the patch and include the center vertex, we have a total of 9 vertices and 8 triangles. This mesh is labeled LOD 1. Using the 4 corner vertices and 2 triangles, we label this mesh LOD 2. Indexing these vertices tells the graphics card which ones are active. A particular patch configuration is called a block and can be of any suitable size (de Boer used a 17x17 block in his implementation).

Geomipmapping relies on a quadtree-based system for culling. A 3D bounding box is created for each block and only those blocks where the bounding box is inside or intersects with the view frustum is included in further calculations. All others can be culled away using a standard quadtree-based traversal.

Based on the distance away from the viewer, higher LOD levels are used to reduce the number of vertices and triangles needed for those blocks. The closer the blocks are to the viewer, the lower the LOD level until full resolution (LOD 0) is used. The challenge of geomipmapping is choosing these distances without causing undoing popping. One answer is to use the same methodology as texture mipmapping. When a mipmap's texel to pixel ratio falls below 1:1, then a lower resolution mipmap is needed. This occurs at a particular distance away from the viewer. Similarly, when switching from LOD 0 to LOD1, we will perceive an error or wrongness to the terrain block due to the removal of vertices which will subsequently change the height of the terrain. This change in height is less noticeable at a distance due to perspective. We can project the change in height to screen space pixels and compare the result against an error threshold. When the projected height change is less than the threshold, the switch is permitted. Despite several possible height changes per terrain block, it is only necessary to perform the check on the vertex with the greatest projected height change. If this value is lower than the error threshold, all the others will be lower as well and thus the block can switch to the lower resolution.

6. Chunked LOD

Thatcher Ulrich's chunked LOD approach is a simple, yet effective terrain rendering technique based on discrete levels of detail. (Polack, 2003; Snook, 2003; Ulrich, 2002) Unlike more complicated CLOD algorithms, its aim is not to achieve an optimal triangulation, but to maximize triangle throughput while minimizing CPU usage. It builds on the notion that current graphics subsystems are so fast that it is usually cheaper to render a few more triangles than to spend a lot of CPU time dropping some unnecessary ones. Of course, level of detail management is still required, but at a much coarser level. The idea here is to apply view-dependent LOD management not to single vertices, but to chunks of geometry.

As a preprocessing step, a quadtree of terrain chunks is built (see Figure 2-20). Every node is assigned a chunk, not just the leaves. Each chunk can contain an arbitrary mesh, enabling mesh optimization during the preprocessing step. The meshes on different levels in the tree have different sizes in world space. The tree does not have to be full, nor balanced, making adaptive detail possible. Each chunk is self-contained (i.e. the mesh and texture maps are packed together), making out-of-core data management and rendering easy and efficient. During rendering, the quadtree nodes are culled against the viewing frustum and visible nodes are then split recursively until the chunk's projected error falls below a given threshold.

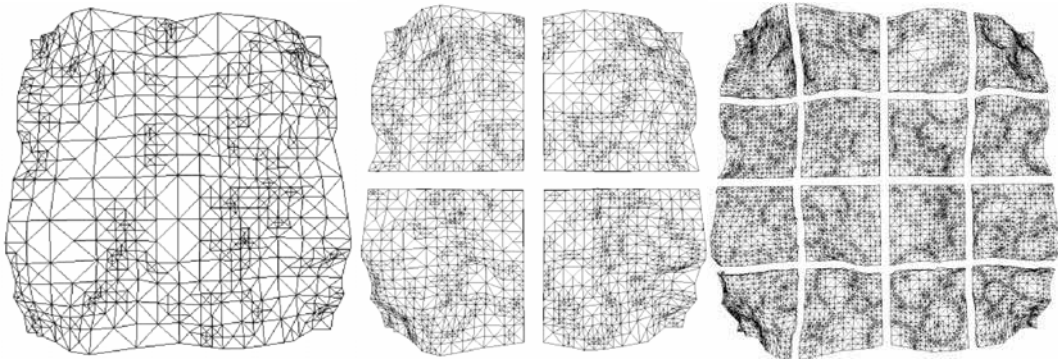


Figure 2-20 First three levels of the chunked LOD quadtree
(From Ulrich)

Since the terrain is assembled from separate chunks, there will be cracks in the geometry between different LODs in the hierarchy. In order to minimize these artifacts, Ulrich uses a technique called skirting. Skirting basically means that on the edges of the chunk's mesh there are extra triangles extending downwards (like the skirting around a bed), effectively hiding the gaps. One must be careful when selecting a skirt height, since long skirts will impact fill rate.

Popping is a noticeable problem with most discrete LOD methods because a lot of the vertices change abruptly when switching between differing LOD resolutions. This disturbing effect is eliminated through distance-based vertex morphing. To do this, Ulrich assigns a delta value to each vertex that tells its distance from the surface of the

parent mesh at that point. With this delta value, it is possible to smoothly interpolate from one mesh to the other.

7. View Dependent Progressive Meshes

Hugues Hoppe extended his original work on progressive meshes (PM) to accommodate large scale terrain rendering. (Hoppe, 1996, 1997, 1998) The idea behind progressive meshes is to construct a special data structure to allow rapid extraction of different LOD meshes that represent a good approximation of an arbitrary triangle mesh.

Building a good PM representation of the mesh is crucial. The process starts with the original high resolution mesh and then gradually reduces its complexity by collapsing selected edges to a vertex. During this mesh decimation, collapse information is saved, making the process completely reversible. The result is a coarse base mesh, together with a list of vertex split operations. The order of edge collapsing is very important to form a good PM. During view-dependent refinement, the active forward-facing vertices are visited and vertex splits or edge collapses are performed based on the projected error.

The main advantage of using a PM is that it supports arbitrary meshes. This means that PM geometry is not tied to a regular grid, and as a result allows PMs to support terrain features such as overhangs, caves, etc. As mentioned before, the challenge here is finding source data to help build such datasets.

The downside of using PMs is that, like chunked LODs, building PMs is computationally expensive and should be done as a preprocessing step. Since the simplification procedure is performed bottom-up, it prohibits processing of large datasets that cannot fit into main memory. The run-time view-dependent refinement procedure is also quite expensive. There is another method called View Independent Progressive Meshes (VIPM) that has the benefits of a good progressive mesh representation without the complicated computations required by the view-dependent refinement. Unfortunately, for large-scale terrains, where view dependent refinement is a must, it is hard to make it useful.

8. ROAM (Real-time Optically Adapting Meshes)

Mark Duchaineau et al. developed an LOD algorithm for use in low level flight simulation called ROAM (Real-time Optimally Adapting Meshes). (Duchaineau et al., 1997; Polack, 2002; Snook, 2003) Since this algorithm was designed largely for flight simulation purposes, a reasonable assumption to make was continuous, smooth camera movement. With no sudden change in one's viewpoint, ROAM exploits frame-to-frame coherence by assuming that the mesh in the current frame and the mesh in the next frame will only differ by a few triangles. If this is the case, there is no need to regenerate the mesh from scratch every frame. This assumption allows the algorithm to maintain a high resolution mesh ("the active cut"), and only change it a little every frame.

ROAM operates on a triangle bintree structure (see Figure 2-21) that allows for incremental refinement and decimation. To perform on-the-fly mesh refinement, ROAM maintains two priority queues that drive the split and merge operations. The queues are filled with triangles and sorted by their projected screen space error. In each frame, some triangles are split for more terrain detail, and others are merged into larger triangles. Since the bintree is progressively refined, it is possible to end the refinement at any time, enabling the application to maintain stable frame rates. In order to avoid cracks caused by T-junctions, the algorithm recursively visits the neighbors of each triangle and subdivides them if necessary (called a forced split, see Figure 2-22). This is an expensive operation, but if only a few triangles need updating then we can afford the extra overhead.

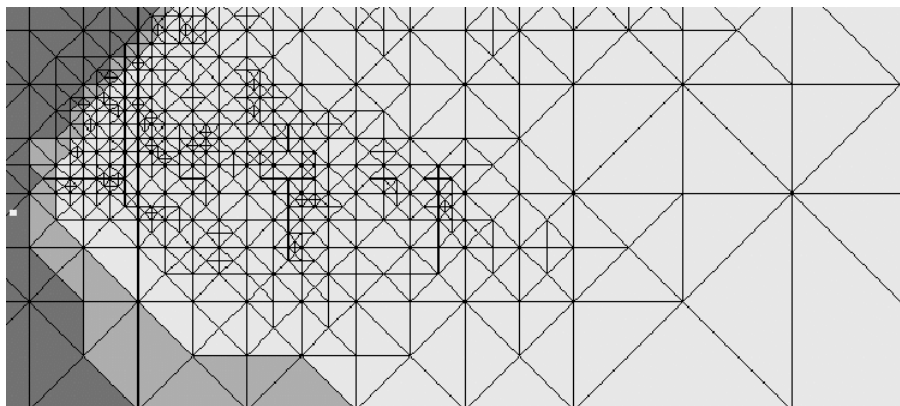


Figure 2-21 ROAM triangulation from an overhead view of the terrain
(From Duchaineau)

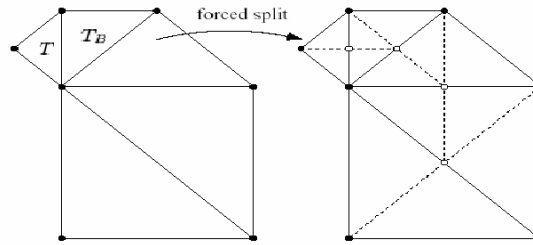


Figure 2-22 Forced split to avoid T-junctions
(From Duchaineau)

Unfortunately, ROAM's biggest flaw as a general purpose CLOD algorithm lies in its initial assumption. For low resolution terrain, the chain of dependency to follow when fixing cracks is not too deep. For higher resolution meshes (e.g. when the viewpoint is near the ground), this operation gets much more expensive. Also, the number of triangles that need updating (split or merged) depends on the relative speed of the camera. It is important to note the term "relative" as camera speed should be measured relative to the terrain detail. Given the same camera movement, the number of triangles flown over (and thus, the number of triangles that need to be updated) depends largely on terrain resolution. If the relative speed is high, the number of triangles that pass by the camera will be high as well. Since most triangles in the high resolution mesh will consist of nearby ones, it means that most of the mesh will need updating. The approximating triangle mesh will be so different every frame that there is little point in updating the mesh from the last frame just to save a few unchanged triangles. Considering how CPU-expensive the update operations are, it is clearly a bad idea to do incremental refinement. This is worsened by a feedback loop effect: as one frame takes more time to render, the camera moves farther, requiring the next frame to change even more of the mesh, that takes even more time to render, and so on.

9. SOAR (Stateless One-pass Adaptive Refinement)

The Stateless One-pass Adaptive Refinement (SOAR) framework was developed by Peter Lindstrom and Valerio Pascucci at the Lawrence Livermore National Laboratory. (Lindstrom & Pascucci, 2001, 2002) It combined some of the best LOD practices published to date and extended them with some very good ideas. SOAR has

many independent components: an adaptive refinement algorithm with optional frustum culling and on-the-fly triangle stripping, smooth geomorphing based on projected error, and a specialized indexing scheme for efficient paging of data required for out-of-core rendering. Since this algorithm serves as the basis for the SOARX algorithm discussed next, it will receive a more in-depth description.

Unlike ROAM, the SOAR refinement algorithm generates a new mesh from scratch every frame. The downside is that SOAR cannot be interrupted in the middle of the refinement. Once SOAR starts a new mesh, it is necessary to finish it in order to get a continuous surface. In reality this is not a serious limitation, because it is possible to adjust the error threshold between frames. Thus, if one frame took too long to construct, the implementation could adaptively change refinement parameters for a quicker build time the next frame.

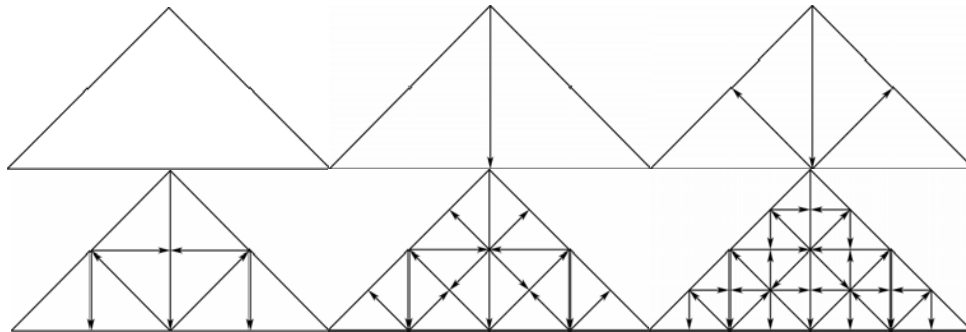


Figure 2-23 Longest edge bisection
(From Lindstrom and Pascucci)

The SOAR refinement algorithm is based on the longest edge bisection of isosceles right triangles (see Figure 2-23); subdividing each triangle by its hypotenuse, creating two smaller isosceles triangles. This subdivision scheme is popular among terrain renderers because meshes can be locally refined (i.e. subdivide small portions of the mesh more than other parts without breaking mesh continuity) and any newly created vertices lay on a rectilinear grid, making it easy to map a surface onto it. Vertices are categorized by the depth of the recursion starting with the root vertex at level zero (i.e. no

parents). Vertices on the edge of the mesh have two children and one parent and all other vertices (save for the leaf vertices) have four children and two parents.

In practice, we typically map our surface onto a square by joining two triangles by their hypotenuse or by joining four triangles by their right apex (see Figure 2-24). The resulting square base mesh can be recursively subdivided to any depth by the longest edge bisection. SOAR recursively subdivides the base triangles one by one until the error threshold is met.

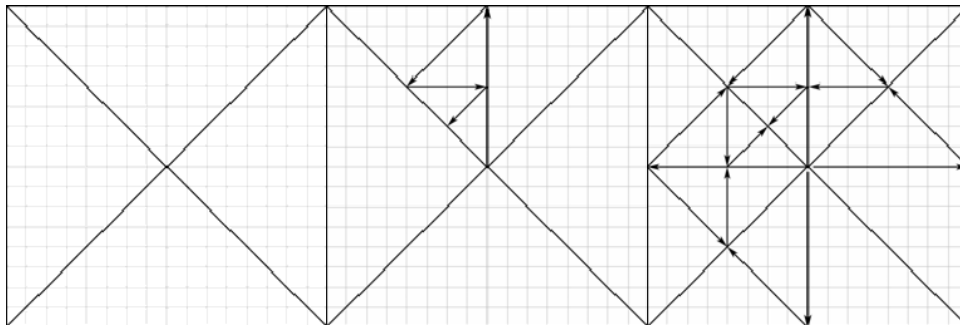


Figure 2-24 Left: joining four triangles forms a square. Middle: refining triangles individually results in cracks. Right: crack prevention
(From Lindstrom and Pascucci)

Although the mesh can be refined locally, this subdivision scheme does not automatically guarantee mesh continuity. In order to avoid T-junctions, we have to follow a simple rule. Lindstrom calls the vertices that are in the resulting mesh “active.” For each active vertex, all of its parents (and recursively all of its ancestors) must be active too. This condition will guarantee that there are no cracks in the mesh. SOAR avoid costly dependency walks by activating (selecting into the mesh) vertices based on projected error metrics and ensuring that these projected errors are nested (a vertex’s projected error must always be smaller than any of its ancestors’). Therefore, if a vertex is active then all of its ancestors will be active, resulting in a continuous mesh. SOAR ensures this nesting property by assigning bounding spheres to vertices (Figure 2-25).

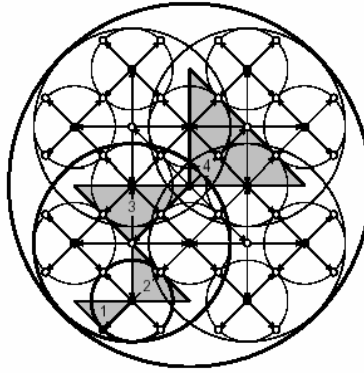


Figure 2-25 Bounding sphere hierarchy

(From Lindstrom and Pascucci)

A vertex's bounding sphere contains all of its descendant vertices' bounding spheres. This nested hierarchy of bounding spheres can be built as a preprocessing step with each sphere being described by a simple scalar value representing its radius. Now, if we project every vertex's object space error from the point on its bounding sphere's surface that is closest to the camera (see Figure 2-26), then the resulting projected error will be nested, thus eliminating cracks.

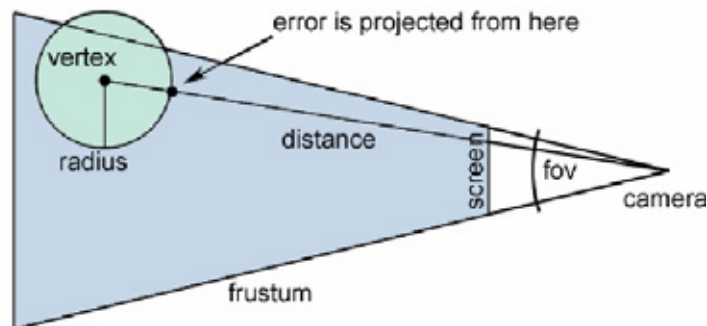


Figure 2-26 Error is projected from the nearest point on the bounding sphere.

(From Balogh)

The nested sphere hierarchy also enables efficient view frustum culling. If a sphere is totally outside the frustum, then its vertex and all of its descendants are culled.

This allows large chunks of unseen geometry to be culled away early in the graphics pipeline. If a sphere is totally inside one frustum plane, then all of its descendants will be inside and further checking against that plane is unnecessary. By keeping a flag for the active planes, we can make sure that only spheres intersecting at least one plane will be checked. The framework also supports a method for smooth geomorphing of terrain vertices, where the morphing process is driven by the vertex's projected error. This is better than time-based geomorphing since it is stateless (i.e. no need to remember which vertices are morphing), and vertices only morph when the camera is moving, making it practically undetectable.

10. SOARX

In 2003, András Balogh, a graduate student at the Budapest University of Technology and Economics, improved the Lindstrom SOAR algorithm with a more efficient refinement strategy, a lazy view-frustum culling technique, and then extended the SOAR algorithm by improving the apparent visual detail of the terrain with procedurally-generated detail maps, distorted displacement maps, and per-pixel lighting. (Balogh, 2003) We will discuss the SOAR extensions in later sections. Here we will focus on Balogh's improvements to SOAR.

Balogh was concerned about SOAR's most performance critical component, the refinement procedure that built a single generalized triangle strip for the whole mesh in one pass. He felt that the simple recursive algorithm used by SOAR was inefficient since it evaluated the same vertices (calculating projected error and performing view-frustum culling) multiple times. His first improvement was to reduce the number of times that a vertex is sent across the bus by using indexed vertices. Resending an index (an integer) is much more efficient than sending the entire vertex data (three floats or doubles), especially when degenerate triangles (a triangle with three vertices on the same line) are needed in the creation of a triangle strip. Balogh optimized the building of triangle strips by minimizing the number of vertices and degenerate triangles required to create a given mesh. Balogh concerned himself with the ordering of visited vertices and how the original algorithm didn't recognize that given a continuous mesh, only certain vertices

and child nodes can be active. By carefully examining the nature of the refinement, it was possible to develop a more sophisticated algorithm that has more knowledge about triangle connectivity and thus produces a more efficient triangle strip.

As discussed previously, SOAR makes use of bounding spheres for hierarchical culling of the view frustum. This is a very efficient method of rejecting large chunks of vertices early in the refinement process. Spheres outside a frustum plane are rejected and spheres inside completely inside the view frustum are accepted without further testing. This method works nicely, but has some subtle problems. First, it is possible to cull away a vertex in the lowest level that was part of a triangle, thus rendering a coarser triangle. This results in popping (most noticeable at the near clipping plane of the view-frustum) and can result in a serious error to the terrain mesh (see Figure 2-27). Balogh correctly notes that this is not acceptable and should be eliminated.

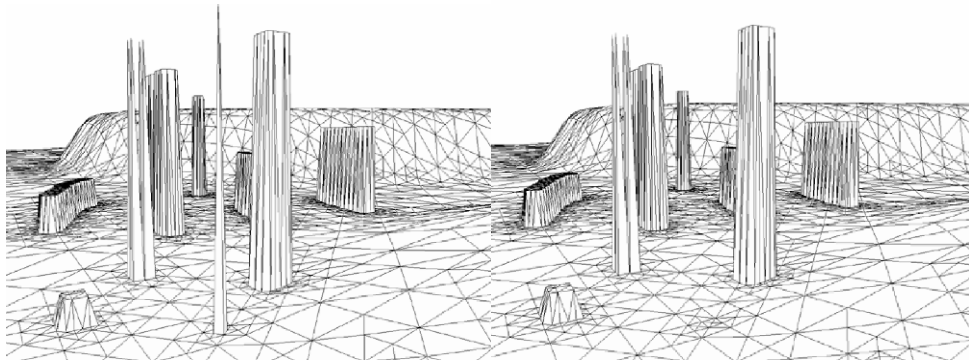


Figure 2-27 Left: The column in the front is inside the viewing frustum. Right: Tilting the camera a bit results in the top vertex of the column being culled by the original algorithm, making the whole column disappear.

(From Balogh)

The original SOAR papers suggested inflating the bounding spheres until they enclose their parents. While culling away thousands of vertices with one sphere is very efficient, culling small numbers (or individual) vertices at the lower levels is very inefficient. SOARX stops culling at the lower levels and accept these vertices automatically as if they were inside the frustum (a.k.a. “lazy view-frustum culling”).

This technique eliminates popping artifacts and results in increased performance since the GPU does a better job of culling individual vertices than the CPU. This technique does not affect gross culling since the majority of vertices are culled away high up in the hierarchy. Only vertices on the lower levels and near the clipping planes are affected. This method has its own problems since as it does not guarantee mesh continuity outside the viewing frustum.

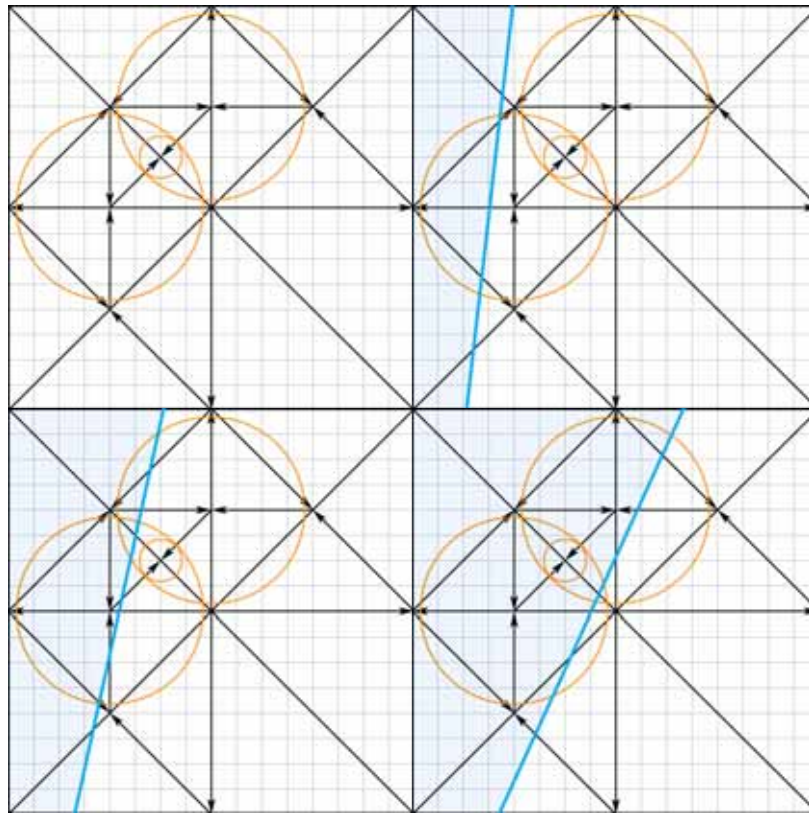


Figure 2-28 Culling the bounding spheres against the view frustum. The shaded area indicates the inner side of the frustum.

(From Balogh)

Examining Figure 2-28, we can see why the new method causes cracks outside the frustum. Assume that the projected error is bigger than the threshold for every vertex (i.e. all vertices are active, unless culled). Using the SOAR algorithm, only the spheres not intersecting with the viewing volume are culled. This means that the vertex with the

smallest bounding sphere on the lower left figure will be culled, coarsening the triangle compared to the lower right figure. This coarsening does not happen on higher levels because the sphere hierarchy is loose enough to enclose the whole triangle. Lazy culling eliminates this popping, since the vertex will be active, even if its sphere is outside the viewing volume (because its parent intersects the volume). Balogh also uses the idea of lazy culling with his method for adding displacement maps (or “dynamic geometry” as he calls it) into the terrain.

J. TERRAIN TEXTURING

So far, we have concerned ourselves with generating and optimizing the geometry of the landscape without paying any attention to the textures that we will apply to this geometry. Thus, we now quickly touch upon simple texture mapping, procedurally generating texture maps, detail maps, texture splatting, and finally, how SOARX made use of textures within its algorithms.

1. Simple Texture Mapping

One of the simplest, and generally least convincing, method to apply color data to our terrain is to take a single texture (like a satellite image) and drape it over the vertex array. The problem with such an approach, first assuming that the aspect ratio of the image matches that of the mesh and the proper texture coordinates have been generated, is that a picture of a landscape contains information that the viewer expects to be in three dimensions (buildings, trees, etc). While a simple texture-mapped terrain may look believable from thousands of feet in the air, it looks entire unconvincing at ground level. Image pixels are stretched thinly across nearby vertices causing a washed-out appearance. Shadows and other lighting information is, of course, static within the image causing the terrain to only look realistic during the same lighting conditions with which the picture was originally taken, thus limiting the usefulness of the terrain visualization. While satellite imagery will be valuable to determine the overall coloring and placement of vegetation within the terrain, it is best to recreate the terrain surface with a procedural texture using the satellite imagery as its basis function.

2. Procedural Texture Generation

There are numerous books, chapters, articles, and reports written on the subject of procedural texturing. (Forsyth, 2002; Milliger, 2002) The most famous of these is the classic text *Texturing & Modeling: A Procedural Approach*.(Ebert, 2003) The upshot of all these works is that textures can be created algorithmically (guaranteeing the ability to reproduce such textures at a later time) and that the majority of these non-repeating, fractal-based textures have a distinctly organic feel to them, making them ideal for use within terrain visualization applications. Procedural textures, like fractals, have unlimited resolution, providing huge texture surfaces with infinite detail. Variety in the terrain's texture can be included and reproduced using noise functions (and PRNGs) as before with terrain meshes. Textures can be influenced by using basis functions like heightmaps and imagery to guide the textures to respect the intentions of the actual terrain features. Finally, while current graphics hardware now supports the ability to easily generate procedural textures at run-time, the more common approach is to pregenerate textures off-line or in tandem with the loading of the terrain's heightmap and imagery data. It is much more efficient to spend GPU cycles performing complex vertex transformations or lighting calculations than regenerating a texture that is unlikely to change during run-time.

3. Detail Maps

Since geospecific imagery or base textures are never high-resolution enough, we can render added high-frequency detail using geotypical textures to avoid having a big blurry surface up close. Detail maps are an easy way to add these minute details to one's terrain through the use of multitexturing. Most detail maps are procedurally-generated high resolution grayscale textures that are repeated (i.e. tiled) many times across the landscape to add apparent nuance features such as cracks, bumps, and rocks. Detail textures are either applied by making two separate rendering passes or by using the multitexturing capabilities found in most modern graphics cards to blend the detail map with the base texture. Blended with a base texture, the detail texture tiles add high resolution features up close to the viewer that would be impossible to obtain without using excessively large base texture maps (eating up tons of memory). The tiling effect is

usually hidden from the ground-based viewer due to perspective, and is typically only discovered by mid and high altitude viewpoints (where it is common to not use the imperceptible detail texture in favor of the dominate base map). Even this effect can be eliminated by scaling and adding detail maps. Daniel Berger combined the scaling and adding of detail texture octaves with alpha maps into a technique he called “spectral texturing.” (Berger, 2003) Presented at SIGGRAPH 2003, he showed on a consumer-level laptop that a convincing real-time high altitude terrain visualization scene could be created with a single quadrilateral and spectral texturing. Not only was this terrain entirely texture-based, it also permitted an “infinite zoom” capability.

4. Texture Splatting

The basic idea of Charles Bloom’s texture splatting technique is to use a few high resolution tile sets to nonlinearly “splat” (i.e. blend localized textures) at run-time versus using a detail texture repetitively over a single (loaded or precomputed) base texture. (Bloom, 2000) Once we have chosen a set of appropriate tiles for a particular region (e.g. grass, dirt, rocks, etc), we want to splat them around the viewer’s immediate area, calculate their influence/weight upon neighboring tiles, crossfade between neighboring tiles, and for those vertices beyond the local area (100-200 meters in a humanoid-based game), resort to a single tile or a base texture. The assignment of tiles in a region should be randomly determined based on the likelihood of those textures appearing within the terrain region (e.g. 50% grass, 30% dirt, 20% rock). Crossfade between tiles can be linear, weighted, or for a more realistic look, use a procedurally-generated noisy alpha map for each source texture’s alpha channel (and then crossfade as before).

5. SOARX Texturing

Balogh’s SOARX algorithms improved upon the original SOAR CLOD algorithms created by Lindstrom et. al. To make the terrain visually appealing, he used multitexturing to combine (among several others) a procedurally-generated detail map with the base texture. Balogh went on to extend SOARX by also using the detail map as displacement map (i.e. “detailed geometry”). Balogh did this to overcome what he saw as a limitation in the original design of SOAR: that while the algorithm could handle

huge, detailed terrain, storage and preprocessing requirements quickly grew beyond the limits of practicality. SOAR could only handle a fixed resolution, static terrain using measured data. Details in the terrain lost to measured data points were also missing within SOAR visualizations. If we only want to display geometry based on accurate measured data, there is nothing we can do and the SOAR approach is acceptable. However, if the added detail is necessary to promote the illusion of reality, the SOAR approach is severely lacking. Usually reality-enhancing details cannot be measured by conventional means (or stored efficiently if they could be measured). Some form of parametric description (soil properties, roughness, vegetation, etc) would make it possible to define different details for different kinds of terrain.

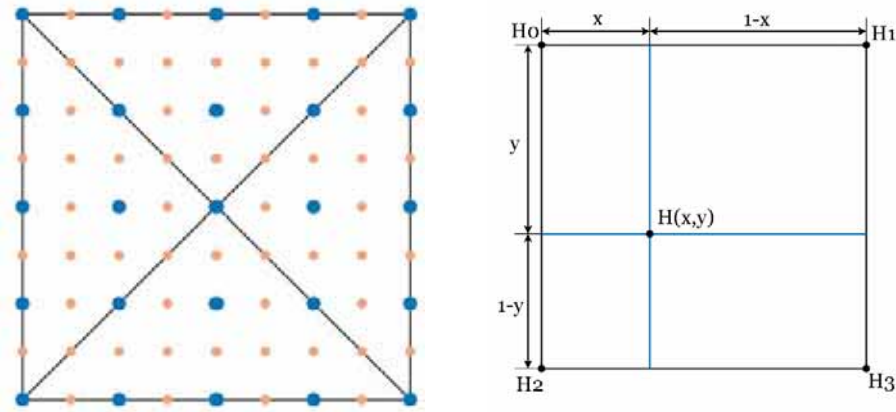


Figure 2-29 Left: embedding detail map Right: bi-linear interpolation
(From Balogh)

Balogh proposes the use of Perlin noise or some other form of procedural detail texture generation or even a precomputed tileable detail texture as a detail terrain map that will be merged with the original base elevation data (the heightmap). However, in order to increase the detail, we have to increase the virtual grid resolution of the terrain. This will also mean adding extra levels to the refinement algorithm described earlier. For example, in Figure 2-29, the original elevation data is shown as larger blue points. The smaller orange points represent the higher resolution of the detail map. Detail values are not inserted directly; instead, a basis function using a bi-linear interpolation between

known elevation postings is computed and the detail values are added (or subtracted) to this basis function with some weighting (e.g. 0% detail contribution on the blue points) to preserve the shape of the original surface. Note that each of these new points will not only need to store its positional information, but also its normal vector (discussed in the lighting section), object space error and bounding sphere radius as well. See Balogh's thesis for details on computing the last two values.

In our example, simple addition was used to add in the heights from the detail map on top of the base map, creating a form of distorted displacement map (which incidentally helps SOARX hide its detail tiling). Note that regular displacement mapping moves vertices perpendicular to the tangential plane of the surface (see Figure 2-30). By using lazy frustum culling and trading bounding spheres for bounding circles (discarding the height component and adding a constant error for each detail vertex), Balogh is able to incorporate the detail/dynamic geometry into the SOARX architecture.

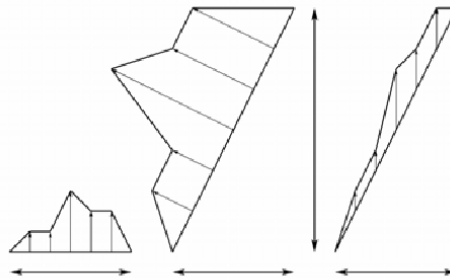


Figure 2-30 Comparison of surface combining techniques (base, displacement, addition)
(From Balogh)

THIS PAGE INTENTIONALLY LEFT BLANK

III. METHODOLOGY AND DESIGN

A. OVERVIEW

This chapter discusses the design of our system based on a functional specification from stereotypical user scenarios and how meeting these user-driven needs leads to a technical specification. A functional specification describes how a product will work from the user's perspective regardless of implementation details. (Spolsky, 2004) It concerns those features that a user requires to accomplish a particular task. A technical specification describes the internal implementation of the system. It concerns data structures, relational database models, programming languages, tools, algorithms, etc. The technical implementation details will be presented in the next chapter.

B. SCENARIOS

In designing a system, it helps to imagine a few real world examples of how actual people would use the system. In the following sections, we examine three usage scenarios and the corresponding expectations of each user with respect to the system. These scenarios assume that GENETICS-based software already exists and that the users are focused on configuring the system to provide appropriate visualization products to their customers. Our users also have at least a basic knowledge of terrain terminology and how to acquire appropriate data to build their scenes.

1. Mission Planning

Lt Rosencrantz is an imagery analyst working for the 505th Intelligence Squadron. His task for today is to evaluate the suitability of a drop zone for a squad of Rangers from the 18th Airborne Corps who will deploy into Czechistan to conduct reconnaissance prior to an air strike on a nearby rebel compound. Maj Player is in command of this mission and needs to quickly see what the terrain looks like from a soldier's perspective. Nobody has been to this remote corner of Czechistan before, so what Maj P. is asking for is some landscape familiarity (i.e. the relationship of the compound to its surrounding terrain) to make a go/no-go decision on the mission.

Lt R. has georectified imagery and elevation data for the region and uses a GIS package to hastily create a simple land cover classification image by manually identifying a few basic vegetation types from the imagery (e.g. trees, bushes, grass) and assigning a unique color to each type. Editing GENETICS' land cover classification (LCC) configuration file, he tells the system about his custom classification scheme and keeps the default ecotype regime settings. Not having vegetation models, he instructs GENETICS to use the default color-matched cones (e.g. green for trees, yellow for bushes, etc), scaling his bushes to be a quarter of the size of a tree. Finished with the LCC specifics, he looks over the system configuration settings. The directory values are already set to the unit's data repository, so all he needs to do is point the system to the proper imagery and his custom LCC image. He also changes the viewer's starting position to the latitude/longitude coordinates of the drop zone. Given that this task is more about data visualization than interactivity, Lt R. feels he can afford to run the program with a high density of vegetation objects (at the cost of reduced frame rate), but the "I need it now!" time pressure forces him to select a low resolution setting for the vegetation placement processing (resulting in a quick startup). Lt R. knows that some aggregation of his land cover image will occur, but as the data was aggregated to begin with he feels the tradeoff in timeliness vs. accuracy is probably worth it.

With Maj P. looking over his shoulder, Lt R. runs the program which creates the scene in just a few minutes. Together they look at the terrain from a variety of locations and lighting conditions. Maj P. is able to see that his planned insertion point will drop his Rangers into a grassy plain with just a few bushes that is within sight of the compound. Based on this information, Maj P. chooses a new drop zone in a clearing on the backside of the southern ridgeline. From this new location, the Rangers can move through the forest to the top of the ridge and spy on the compound below.

2. Mission Rehearsal

Capt Guildenstern is Maj P.'s simulation operations officer and is responsible for teaching the Ranger squad about their new mission plan. Lt R. has provided Capt G. with all the source data and configuration files he used in his mission planning analysis.

Additionally, Lt R. found actual land cover classification data of Czechistan from the CORINE Land Cover 2000 Project. Lt R. reconfigured the LCC configuration file to reflect the color values of the 44 different classes of CORINE data. Capt G. selects six LCC types and matches them up to his in-house inventory of European vegetation models. Based on his knowledge of the Southern European theater, Capt G. is able to refine the LCC regime settings to better match vegetation growth in that region.

Since much greater detail and accuracy is needed to conduct a mission rehearsal exercise, Capt G. chooses a smaller region than Lt R. to concentrate on. With this smaller region, he can increase the resolution of his vegetation maps, resulting in a closer match to real-world growth patterns. The smaller region also allows Capt G. to maintain an interactive frame rate which he uses to move beyond simple terrain visualization into the realm of building a realistic small-team training environment.

Capt G. configures each player workstation with the same data and settings. At run-time, his networked dismounted infantry simulators are able to generate identical representations of the environment without the need for a pre-built database. On the second run-through of their mission rehearsal exercise, Capt G. notices a couple of his Rangers are beginning to memorize the scenario by running blindly over the ridgeline to a grove of trees on the other side that they know provides excellent cover. To force his training audience to be more cautious in their movement tactics, Capt G. instructs the players to increment the system's random number seed each time through the exercise. Doing so prevents vegetation objects from appearing in exactly the same location and orientation from run to run, although the overall look of the terrain does not change.

3. Large-Scale Joint Exercise

Col Hamlet is responsible for training the joint tactical operation against the rebel strongholds in Czechistan. To that end, he instructs his simulation chief, LtCol Horatio, to create a large-scale joint exercise based upon the same scenario (i.e. source data, configuration files, and model library) as Capt G. and the Rangers from the 18th ABN Corps.

With a region of interest larger than a geocell, LtCol H. decides to lower the resolution of the vegetation maps and reduce the vegetation density to a point where all participants can maintain interactive frame rates. LtCol H. has obtained a detailed geospecific model of the compound for this exercise. Vegetation growth within the compound is prevented by modifying the original land cover image with a color-map mask for that portion of the terrain. Each participant in the networked virtual environment is able to see the same landscape (terrain mesh, surface texture, and vegetation object placement), allowing units to coordinate or conceal their operations based on landscape features.

C. DESIRED FEATURES

Based on our scenarios, a user of GENETICS will interact with the program largely through the gathering of input data and setting variables within configuration files. The system and land cover classification configuration files should be easy to edit and in a user-readable format. Missing whitespace and settings or improper formatting should not cause the system to crash. Editing a configuration file should be as easy as editing a simple webpage. Thus, we will use the Extensible Markup Language (XML) to simplify both the editing and parsing of configuration files. XML tags should have clearly defined names to minimize errors and reduce specialized training on the system. For the land cover classification XML file, we need tags for each LCC type that include:

- Unique index value
- Color map values in red, blue, and green
- Descriptive name (e.g. “marshland”)
- Topographic regimes to include appropriate ranges for elevation, relative elevation, slope, and aspect angle.
- List of geometric models with appropriate scaling for each

For the system configuration XML file, we will need tags that include:

- Directory locations of elevation source data
- Filenames of the satellite and LCC images
- Starting location in latitude/longitude coordinates
- Setting the accuracy/resolution of the vegetation maps
- Setting the density of the vegetation
- Random number seed to control variability/repeatability

Occasionally, users will want to either create or modify a source or cached image using a GIS tool or paint program in order to mask out a particular region from object placement (e.g. to remove procedurally placed objects from a region that will be filled with a geospecific inset model or to reduce the overall size of a playbox to increase vegetation density, accuracy, or complexity) or generate their own custom-built LCC scheme. GENETICS needs to support masking and importation of a wide variety of LCC schemes.

D. NON-GOALS

It is likewise important to state what GENETICS will not do (in addition to those stated in Chapter I). In this software version, we will not support the following features:

- Vegetation object creation
 - Assumes the user has access to a model library or uses the default models
- Texture splatting (Bloom, 2000)
 - Assumes the use of imagery or pseudo color ramps based on elevation

E. DISCUSSION

GENETICS is designed to be a terrain visualization component of a larger system, be it a game/simulation engine or GIS tool. For GENETICS, the procedural

techniques to manipulate the geometry, the surface textures, and the vegetation objects must all work in perfect harmony. This synergy must be accounted for in addressing a common problem within data visualization: how to add details/data where none exist?

Let us take as an example, a simple 2D case where two consecutive known elevation postings are 100 meters apart. Their measured elevations are 100 meters and 200 meters respectively. Obviously there exists an infinite number of possible paths that can be drawn between these two points in space. The simplest solution, linear interpolation, is a single straight line going between the two points in a 45 degree angle with respect to the horizontal plane. The 3D case, bilinear interpolation, is the most commonly used technique in creating terrain meshes. The resulting mesh, while quick to calculate and easy to implement, looks like the facets of a gemstone at ground level. Perhaps a more realistic profile is one where a sharp cliff separates two gently sloping plains. This profile could come as the result of imagery analysis or letting Perlin noise (Perlin, 1985) disturb what would otherwise be a long, steeply sloped polygon. Perlin noise functions output the same number each time given the same input, allowing our enhanced terrain mesh to stay constant from run to run if desired, or to create a new mesh (albeit based on the same original data) by simply changing input parameters. Given our noise-enhanced elevation mesh, our algorithm must be smart enough to generate a corresponding detail map that accounts for topological variations caused by the newly generated elevation points.

Continuing our example above, we find land cover data that classifies one plain as grass and the other as deciduous trees. Our algorithm blends ecotope zones to create a realistic transition between dissimilar zones; however, our deciduous trees should not be present on the cliff-side in an equal probability as on the plain. Tree growth is hampered by the harshness of certain environmental factors (e.g. unprotected, steep, inability to retain water) and promoted within other regimes. One can sense the need for topological properties being used to control the appearance of the ground and what objects are most likely found within those environments. This is the point of GENETICS.

Since this terrain is being generated and enhanced for a simulator with limited processing power, a level-of-detail scheme is utilized to display the environment almost

identically (within perceivable limitations) regardless of the type of simulator. Thus, a tank driver may be only able to see the edge of a forest; the helicopter pilot may see the edge of the forest and on to the top of the first major ridgeline; and the aircraft pilot may see a vast forest stretching for miles over hilly terrain. This is an example of aerial perspective and it determines how far the viewer can see in his environment (noting that distant objects will appear less detailed than close objects). Looking at Figure 3-1, our tank driver is able to distinguish the branches of a tree, but not its individual leaves. The helicopter pilot can see each tree, but not necessarily individual branches. Finally, only a low-flying, slow-moving aircraft pilot will be lucky enough to recognize individual trees. For each reduction in detail of a particular object, more objects can be added to the scene. There exist numerous schemes to control the level of detail for objects within a scene. (Luebke, 2003) A common solution to use models with multiple object representations under an LOD node. Thus, for close inspection of an object, full geometry is used while distant objects are represented with a single billboard. Additionally, multiple objects can be spatially grouped together and partitioned within a scene graph using an efficient data structure (e.g. a quadtree) with leaf nodes that are themselves LOD nodes. Thus, an entire group of objects can be “turned off” as the viewer moves outside of the range of the LOD node.



Figure 3-1 LOD and aerial perspective (for tank, helicopter, & airplane)

As shown in Chapter II, the ability to optimize scene complexity of terrain surfaces has been well researched and several popular continuous level-of-detail (CLOD) algorithms exist. We have chosen the SOARX scheme for the purposes of this research project. With the addition of noise to create details between the known elevation

postings, we can use the SOARX algorithm to tessellate the original triangle mesh with the added noise data into many smaller triangles around the player's position. Obviously, this additional tessellation step won't need to happen for the aircraft pilot and will infrequently happen for the in-flight helicopter pilot since they will be unable to perceive the level of detail generated. Since their view of the terrain requires them to see objects on the distant horizon, their terrain polygon budget is best put to use in creating a more expansive environment. For the tank driver or the infantry soldier, this added detail will serve to make combat and movement much more interesting, allowing for the creation of rises and falls in the terrain that were previously unavailable in large-scale simulation environments.

For vegetation generation, we need to process land cover classification images that are either created manually through imagery analysis or downloaded from an authoritative data source. Pixel by pixel, RGB color data will be examined in these images to identify particular LCC types and then compared against topographically derived data of the region in question. From these values, a probability map of each ecotope (i.e. LCC type) can be generated to help build the terrain's ecosystem. Ecotope characteristics (e.g. regimes) are defined in advance and placement is a function of the above topographic parameters in a manner similar to Hammes' ecotyping modeling scheme (Hammes, 2001). Drawing random numbers against each ecotope's probability map determines the composition of vegetation objects that exist within each atomic region (i.e. the terrain surface corresponding to a single pixel within the image map). There needs to be a mechanism to blend multiple ecotopes within the same atomic region to account for transitional zones between largely homogenous vegetation zones. This can be accomplished by decomposing the composite LCC image into a black and white (i.e. "hit" or "miss") image for each ecotype and then smoothing each image with a filter that spreads the probability of placement beyond the original picked pixel. Careful selection of an appropriate filter can emphasize the likely density of ecotopes within a particular region. Seeded random number streams will allow users to recreate landscapes as necessary; ensuring that a tree or bush created by one player's GENETICS system will be

created by all players within the simulation (as necessary depending on their level of detail needs).

To conduct an networked exercise, source data and configuration files must be distributed (or accessible via a shared repository) to each host prior to execution. The initial GENETICS implementation will not support dynamic manipulation of the terrain since doing so would require a terrain server to log all changes to the environment and send this information to any late joiners. (Singhal & Zyda, 1999) Despite not using a terrain server, terrain consistency between players is maintained by using GENETICS with the same source data and configuration parameters. Thus, given a line-of-sight calculation, the results from either observer or target player should be the same since the terrain is generated consistently using the same algorithms across both platforms. Boundary conditions will still exist (e.g. a small fall in the noise-enhanced terrain causing a prone infantry soldier to believe he is hidden from an entity that wouldn't be able to render infantry-level details), but for the vast majority of situations, GENETICS-generated terrain should be perceptibly the same and functionally similar.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION

A. OVERVIEW

The goal of our research is to replace the barren landscapes found within most 3D combat simulations with detailed terrain and natural surroundings that dramatically increase both the realism and difficulty of the training environment. We posit that there are many unmet visual cue requirements (e.g. vegetation clutter) within existing simulators that are vital to the effectiveness of simulator-based training. Our approach enhances the apparent quality of the given set of terrain elevation data and surface imagery, adds vegetation objects that are placed similarly to the arrangement within the actual environment (better than most standard geotypical techniques), and generates a plausible synthetic terrain environment where data is missing or incomplete. Additionally, we wanted to simplify the process of constructing landscapes so that all that was required was to gather source data, edit a simple text file, and run the simulation. Terrain is created by GENETICS within the simulation application at run-time without the need of a precompiled terrain database. This chapter describes the implementation details needed to satisfy the functional specification from Chapter III.

B. GATHERING DATA AND CREATING MODEL LIBRARY

Creation of a source data repository and model library is not expected to be a typical user task since it is likely this data is an already existing shared resource within an organization. Regardless, this process is included for completion and for those users who will need to create such repositories from scratch.

1. Organizing Elevation Data

A request for Digital Terrain Elevation Data (DTED™) from the National Geospatial-Intelligence Agency resulted in the exchange of hard drives where the data was organized in a fashion whereby each DTED™ level was separated into its own directory which was further subdivided by longitude with files specified by latitude (see Figure 4-1). Within GENETICS, the DTED™ path is “c:\dted\level1” and the algorithm conducts searches based on longitude-named directories and latitude-named files.

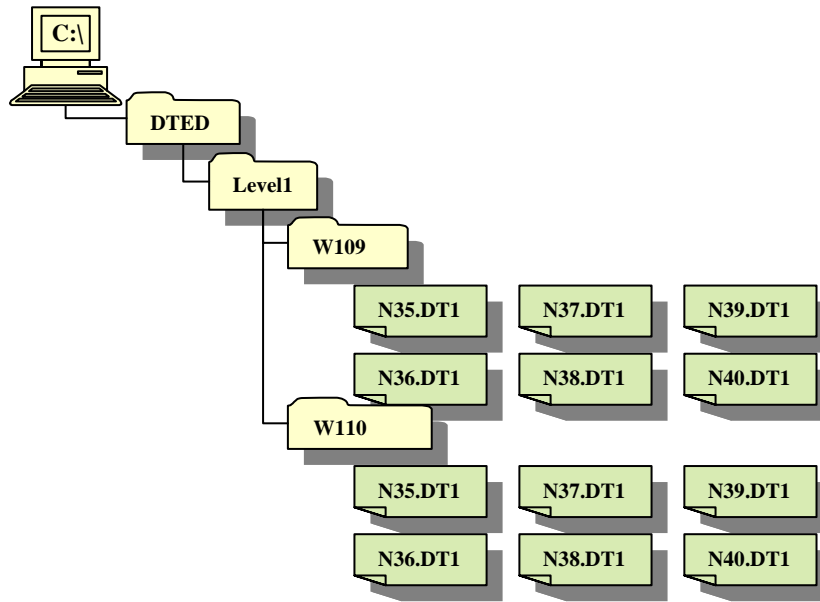


Figure 4-1 DTED™ directory structure

2. Locating Imagery and LCC Data

While it is assumed that imagery of the region of interest will be available to the user, we record here how we obtained our test data. The satellite imagery used for our ground texture was downloaded from the California Spatial Information Library (<http://gis.ca.gov/>). Specifically, we downloaded multiple LandSat 7 GeoTIFF images from the CalView library at <http://casil-mirror1.ceres.ca.gov/casil/gis.ca.gov/landsat7/>. This repository organizes images by path/row (see Figure 4-2) and then by the date of the image. Within each dated (year, month, day) directory, we find multiple images corresponding to various Thematic Imaging band combinations. The closest match to visual perception are the “321” images where band 3 is represented with red color values, 2 in blue, and 3 in green. We selected the best images (e.g. low amount of clouds) for our test region, but in one case the seasonal color variation was too great in the cloudless image and cloudy image was chosen to replace it. We arranged the loading of these images in the system configuration file such that where possible, we would have cloudless images overlap the cloudy one.

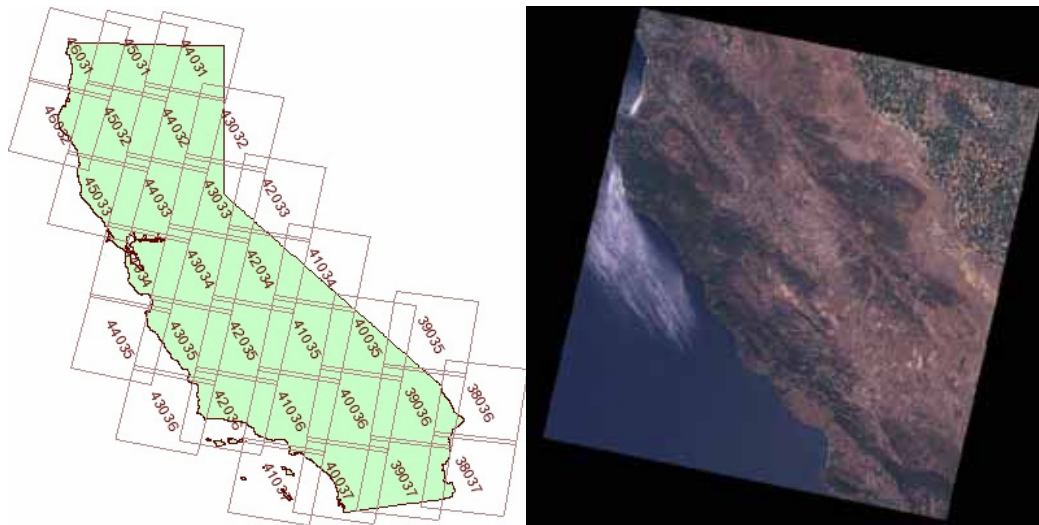


Figure 4-2 CalView path/row numbering scheme and image from path 43 row 035

The USGS Seamless Data Distribution System (<http://seamless.usgs.gov/>) offers users the ability to download NLCD images from the Internet (see Figure 4-3). The user selects an area and then the LCC data can be downloaded as a GeoTIFF. Note that for custom-made or non-U.S. locations, any GeoTIFF-based LCC scheme can be used.

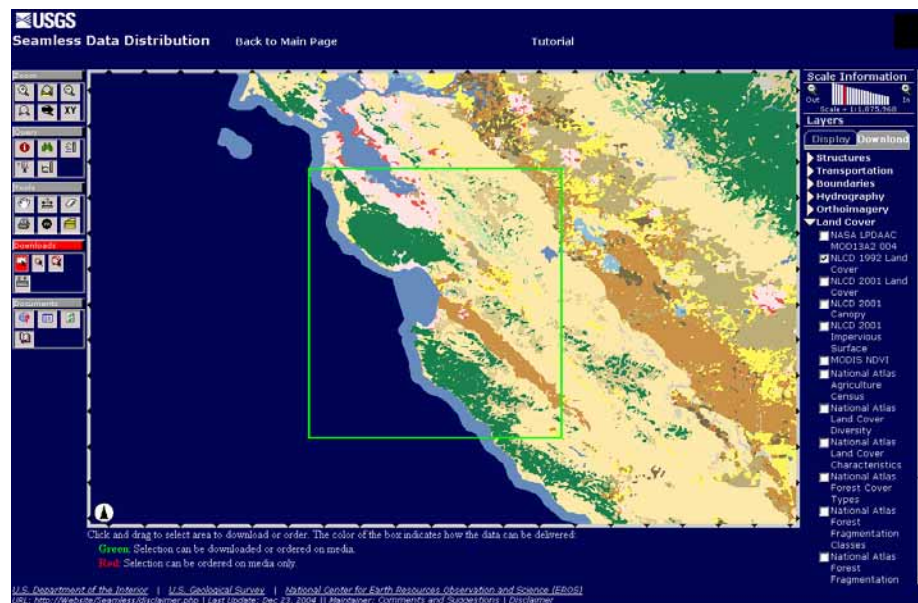


Figure 4-3 USGS Seamless Data Distribution Server

3. Building a Vegetation Model Library

GENETICS, through its use of Open Scene Graph, is able to import a wide variety of visual model formats (3dc, 3ds, flt, geo, iv, ive, lwo, md2, obj, and osg). For example, our placeholder cone objects are stored in a 3ds (3D Studio®) format while our high fidelity models make use of the flt (OpenFlight®) format. Mixing model formats within GENETICS is supported. OpenFlight® models can be created with multiple LOD nodes. As a viewer moves away from the object, the model switches from a geometric representation to a cross-polygon model to a billboard (see Figure 4-4). Building a variety of realistic, multiple LOD plant models was accomplished using a commercial package (i.e. REALnat® by Bionatics®). As stated previously, it is our belief that most users and organizations have already invested in building a model library. We had not yet created such a library for vegetation objects, so in order to quickly generate these detailed vegetation models, we used a commercial tool as there are currently no comparable open source tools available.

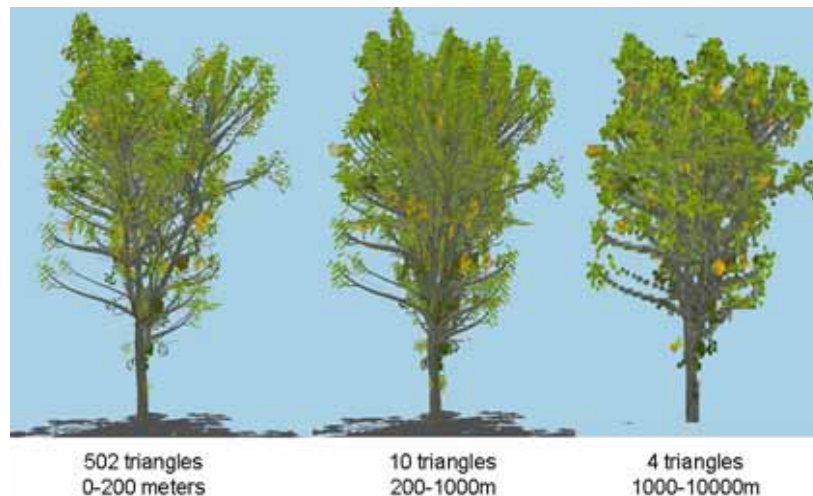


Figure 4-4 OpenFlight® LOD model created with REALnat®

C. CREATION AND PARSING CONFIGURATION FILES

XML configuration files were implemented to allow a user to make significant changes to their terrain synthesis process without needing to recompile GENETICS or their application. Several of the values in this file have a 1:1 correspondence to key

variables within the SOARXTerrain class. In the following sections, we will look at the contents of the soarxterrain.xml and lccdata.xml configuration files.

1. The soarxterrain.xml System Configuration File

Below is an example of a soarxterrain.xml file. The first line specifies where the cache path is located. This is where image files (e.g. heightmap, probability map, etc) and mesh connectivity data are stored for later use (i.e. for runtime processing and for speeding up subsequent runs using the same terrain parameters). GeoOrigin sets the origin (0, 0, 0) of the world coordinate system in geodetic terms. DTED path sets the starting point for elevation data searches (as described previously). Alternatively, a DEM (Digital Elevation Model) path can be given, but due to the unique narrative format of DEM filenames (e.g. Monterey-W.dem, Santa_Cruz-E.dem), this string must point to a specific filename versus a directory structure. GeospecificImage filename captures the GeoTIFF image file(s) that will be used for creating ground textures. MaxTextureSize sets the resolution of both the ground texture and the LCC images (if being used). Currently, only the values 1024 (“low”) and 4096 (“high”) are supported. For a 100km x 100km geocell, this equates to physical pixel sizes of roughly 100m and 25m respectively. The Gamma Correction value determines how much to lighten a dark geospecific ground texture. The LCCImage filename identifies the LCC source image to use and sets a flag to load and parse the lccdata.xml file. Random Seed is used to set, and thus guarantee, the same stream of random numbers is used to determine vegetation placement within each geocell. Looks Per_Pixel sets the maximum number of random draws per pixel per vegetation type. The actual number of looks is modified downwards based on the underlying terrain’s aspect angle in relation to the preferred aspect angle of the particular LCC type in question (see next section on the defining the LCC configuration file). CellMax Objects sets an upper limit for the number of objects of a particular LCC type can be placed within a geocell. Note in our example that the road settings have been commented out. This capability of XML parsing allows one to store multiple settings or make notes about particular settings within the file without parsing these statements.

```

<SOARXTerrain cachePath="cache">
  <GeoOrigin latitude="36.58" longitude="-121.85" elevation="0.0"/>
  <DTED path="c:/dted/level1"/>
  <GeospecificImage filename="43035.tif" />
  <GeospecificImage filename="43034.tif" />
  <MaxTexture Size = "1024"/>
  <Gamma Correction = "1.7"/>
  <LCCImage filename = "mb_lcc.tif" />
  <Random Seed="42" />
  <Looks Per_Pixel="16" />
  <CellMax Objects="2000000"/>
  <!--
  <Roads filename="TGR06053"
    query="SELECT * FROM CompleteChain WHERE CFCC LIKE 'A%'"
    width="8.0"
    texture="road.bmp"/>
  -->
</SOARXTerrain>

```

2. The lccdata.xml LCC Configuration File

Below is an example of a lccdata.xml file. A user of GENETICS records the color-mapped values (in red, green, and blue) assigned to each LCC type (i.e. “Definition Index”) of their particular LCC dataset along with a short narrative description in the lccdata.xml. Also included in this file are the topographic regimes for each LCC type and the various geometric object models that will be placed in the scene corresponding to each LCC type. The topographic regime characteristics include both minimum and maximum values and a sharpness factor that accounts for the exponential value in the influence curve (where “1” represents a linear slope and “2” a square exponential slope). Currently, we are using only a linear influence model for all our regime characteristics, but the GENETICS algorithm does support the use of nonlinear curves. The preferred aspect angle directs GENETICS to promote growth in this slope direction and restrict growth in the polar opposite direction. Multiple object models can be assigned to a single LCC type (e.g. differing species and/or young, medium, and old versions of a single species) and models must be findable by GENETICS to be considered valid. For GENETICS to consider a LCC type valid, it must have a complete Definition and at least one valid Model. Thus, in our example, the water LCC type is not valid since it has neither a complete Definition nor a single valid Model associated with it. Note also that each model can be assigned its own scaling factor.

```

<LandCoverClassificationData>
  <LCCType>
    <Definition Index="11" R="110" G="130" B="177" Name="water" />
  </LCCType>
  <LCCType>
    <Definition Index="43" R="212" G="231" B="177" Name="mixed forest"
      SlopeMin="0" SlopeMax="45" SlopeSharpness="1.0"
      ElevMin="0" ElevMax="3000" ElevSharpness="1.0"
      RelElevMin="-127" RelElevMax="45" RelElevSharpness="1.0"
      Aspect="225" />
    <Model Name="Chestnut.3ds" Scale="0.9"/>
    <Model Name="Sweet_gum_1.flr" Scale="1.1"/>
    <Model Name="Sweet_gum_2.flr" Scale="1.1"/>
    <Model Name="Coastal_Oak3.flr" Scale="1.3"/>
  </LCCType>
</LandCoverClassificationData>

```

D. PROCESSING ELEVATION DATA

Our algorithm begins in earnest by processing elevation data points to create 1 degree by 1 degree skirted height field meshes of the terrain. Height field data is imported directly from an elevation data repository (e.g. DTED™) at run-time. Ground surface details between the known elevation postings are added by subdividing the base mesh and increasing or decreasing the values of the linearly interpolated midpoint heights with Perlin noise. Filtering the amplitude of the noise based on elevation values (e.g. larger elevation changes possible in mountainous settings versus plains) helps to overcome the appearance of randomness between postings. Using the SOARX continuous level-of-detail (CLOD) algorithm (Balogh, 2003), we take our enhanced height field data and construct a dynamically optimized mesh grid based on the user's view frustum. As the user nears the edge of the terrain, we determine the next geocell's coordinates, load up the corresponding source data from our repository, and process the next 1 degree by 1 degree geocell in the same manner. These techniques allow us to offer the user a nearly endless supply of optimized elevation meshes derived from raw source data with increased (albeit artificially generated) resolution over the given source data.

Over our elevation mesh, we drape satellite imagery shaded at run-time with normal maps (i.e. one for the base gradient and another for the detail gradient) to add relief shading and surface details corresponding to the noise-generated additions to the elevation data. A geocell's height field data, noise data and associated textures are

cached for improved load times or can be regenerated afresh if desired. The ability to consistently recreate the same mesh every time is controlled by user selection of a random number seed.

E. CREATING TOPOGRAPHIC IMAGES

At this point, we have created a terrain visualization environment (i.e. imagery over a mesh) that could easily be used for many flight simulation applications. However, in order to bring in participation from ground-based and low-level aerial vehicles, we need to look at adding more details (e.g. large amounts of “geosimilarly”-placed vegetation) to our landscape. With the speed of today’s graphics hardware, the automated placement of millions of objects (vs. rendering them) was seen as the greatest obstacle to creating believable landscapes. In order to properly match our objects’ placement to topographic features within the environment, we need to create height maps, slope maps (with aspect angles), and relative elevation maps using our geocell’s height field data (see Figure 4-5). The reasoning for creating each image will be described shortly, but for now it is suffice to say that they are needed for the GENETICS vegetation placement process.

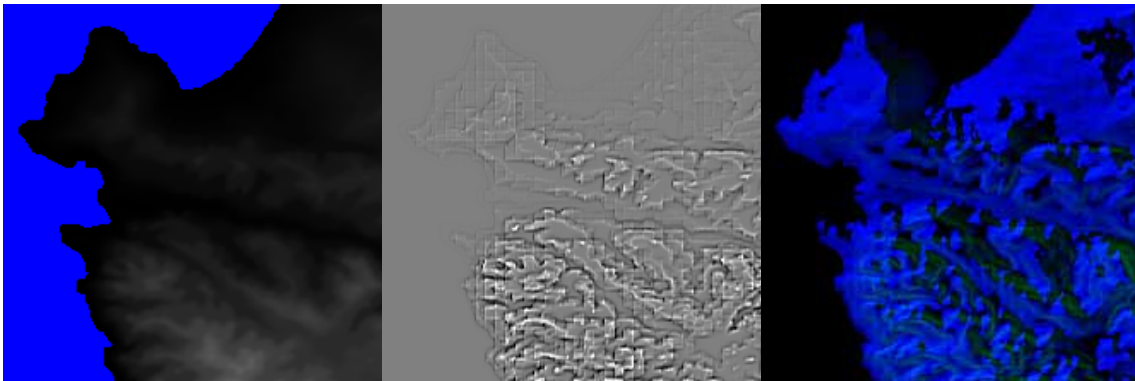


Figure 4-5 Height map, relative elevation map, and slope map with aspect angle

F. CREATING LAND COVER PROBABILITY MAPS

After loading the LCC source image into the system, we create a black and white “picked points” image for each desired LCC type where black pixels correspond to those pixel locations in the composite LCC image that match the color value of the selected LCC type. A union of all of these “picked points” images colored with their respective LCC color values would recreate the original composite source image. This property demonstrates the “all or nothing” approach of using LCC images. Each pixel (and thus its corresponding ground location) is designated a single LCC type with no overlap possible. Since this situation rarely occurs in nature (particularly in transition zones), we need to “smooth” this data from black or white (i.e. on or off) to various shades of gray (i.e. probabilities of occurrence). Alternatively, we could mix our “picked points” images with other LCC images with density values (e.g. NLCD’s Imperviousness or Tree Canopy data), but doing so will still not create the desired overlap between LCC types.

		3		
	2	1	2	
3	1	X	1	3
	2	1	2	
		3		

Figure 4-6 Third nearest neighbor weighting scheme

We create a smoothed image for each LCC type using the “third nearest neighbor” weighting scheme used by Gergel and Turner (see Figure 4-6). (Gergel & Turner, 2002) If our current pixel is a “picked point” (i.e. a “hit”) for that LCC type, it earns a score of 50. A “miss” earns no score. We then look to the north, south, east, and west of our current pixel. A hit from any of these four pixels earns our current pixel another 6.82 points each. From our next nearest neighbors, the four diagonals (i.e. northeast, southeast, southwest, and northwest of our current pixel), each hit earns another 3.41 points. Finally, for our third nearest neighbors, the pixels beyond each of our nearest neighbors in the four cardinal directions, we earn a 2.27 for each hit. With this smoothing function, a “hit” pixel located in a dense patch of other “hit” pixels will

trend towards a score of 100 while an isolated “hit” pixel will trend towards 50. Likewise, a “miss” pixel that is completely surrounded by “hit” pixels will trend towards 50 while a “miss” pixel far removed from any “hit” pixels will trend towards 0. Using this filtering scheme, we have created an initial probability map for each LCC type by relating a pixel’s composite score to the blackness or whiteness of each pixel (see Figure 4-7). However, we need to manipulate this map further to account for topological influences that we derived previously.

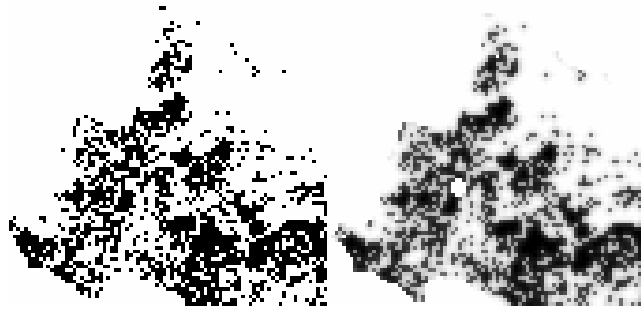


Figure 4-7 Picked points vs. smoothed with masking

A masking image adds the capability to prevent a specific LCC type or multiple types from occurring in a particular region. This allows for easy removal of vegetation objects from lakes or recent alterations to the terrain (e.g. defoliation, clear-cutting, wildfires), but can also be used to designate an area on the map where a geospecific urban environment needs to be placed. Naturally, the masking image can take on the form of either a “picked points” image or a smoothed image. In the “picked points” case, “hits” on the mask will result in deleting any values in the corresponding LCC smoothed images. The resulting LCC smoothed images appear harshly “cut” with the masked image. This approach is good for setting aside rectangular areas for subsequent insertion of detailed geospecific models. Of course, this type of masking could be applied to the earlier LCC “picked points” images and then smoothing would proceed as normal, resulting in some incursions of LCC object within the masked area. In the case of using a smoothed masking image, a more gradual reduction in values occurs against the smoothed LCC images. This approach is good for negatively influencing LCC types against a natural phenomena (open water, lava flows, mudslides, forest fires).

The idea of using topographic features to affect vegetation placement was inspired by Hammes' work on ecotope modeling. The value of each pixel (i.e. the probability of vegetation placement) from an LCC's smoothed image is increased or decreased based upon the preferred regimes of that particular LCC type. We have chosen linear scaling for manipulating probability values, but other scales (e.g. exponential) may be more appropriate for a particular region or influencing factor. GENETICS allows for experimentation in this regard through its use of "sharpness" parameters for each LCC regime characteristic. Each regime characteristic defines a range of topographic values where one is likely to find the particular LCC in question. For example, deciduous trees are unlikely to grow on a very steep slope. Our `lccdata.xml` LCC configuration file specifies a maximum slope angle and probabilities are negatively adjusted as we approach that angle. Aspect angle (i.e. the direction of slope) will affect the density and growth of some LCC objects. For example, in the North Hemisphere, trees typically grow better on slopes that face south or west, resulting in denser, older growth. (Hilts & Mitchell, 1999). Evergreens may have a maximum elevation regime of 3000 meters, which means it is highly unlikely an evergreen will grow (i.e. should be placed by the GENETICS algorithm) above that value. This allows for creation and/or enforcement of timberlines. Relative elevation (i.e. the difference between a particular point's elevation and the average elevation of its neighbors) is an effort to recognize that dips and ravines in the terrain are likely to receive more water and be more sheltered from the weather than rises and ridgelines. Thus, we may wish to bias our probability map to promote vegetation placement within valleys (i.e. negative relative elevation) and reduce the probability of vegetation placement along ridges (i.e. positive relative elevation).

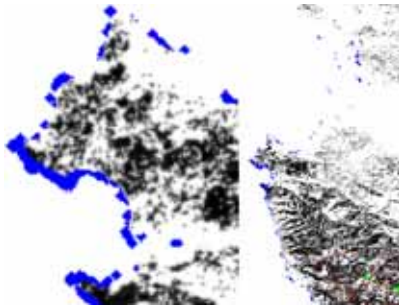


Figure 4-8 Evergreen probability map

Each of the above factors contributes to the final score of each pixel for each LCC type. Collectively, these pixels form our final probability map reflecting the likelihood that an LCC type exists within a pixel's corresponding area on the terrain surface. In Figure 4-8, grayscale values correspond to evergreen placement probabilities; blue pixels indicate locations where potential placement was barred as the area was deemed too low for vegetation growth (similarly, green pixels are too high and red pixels are too steep). Random draws against these probability maps determine the type, location, density, and appearance of the vegetation objects found within the synthetic natural environment (see Figure 4-9). Randomized orientation and scaling of the objects give the appearance of a greater variety of models.

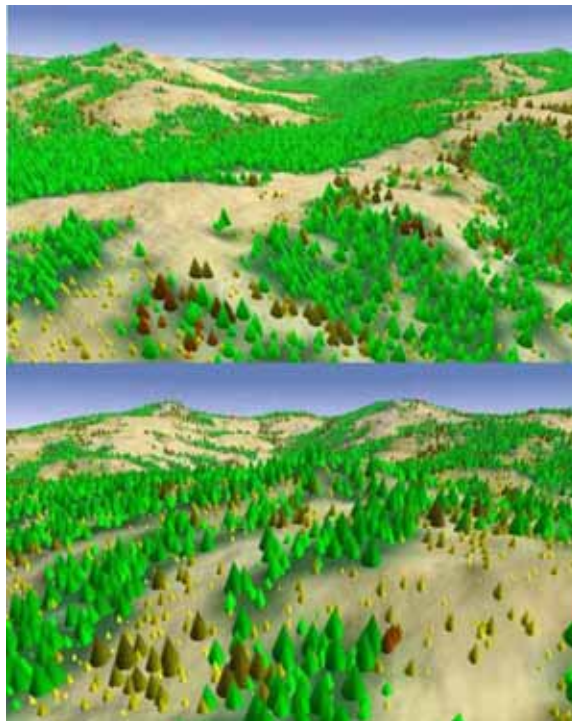


Figure 4-9 Examples of GENETICS vegetation placement with placeholder objects

Using LCC-appropriate vegetation models, the resulting procedurally created geotypical distribution looks realistic (see Figure 4-10) with overlapping vegetation types occurring naturally within transition zones. This simple algorithm can also be extended to incorporate soil moisture or other factors (e.g. prevailing winds, proximity to water) or to generate geotypical distributions of man-made landscape features (see Figure 4-11).

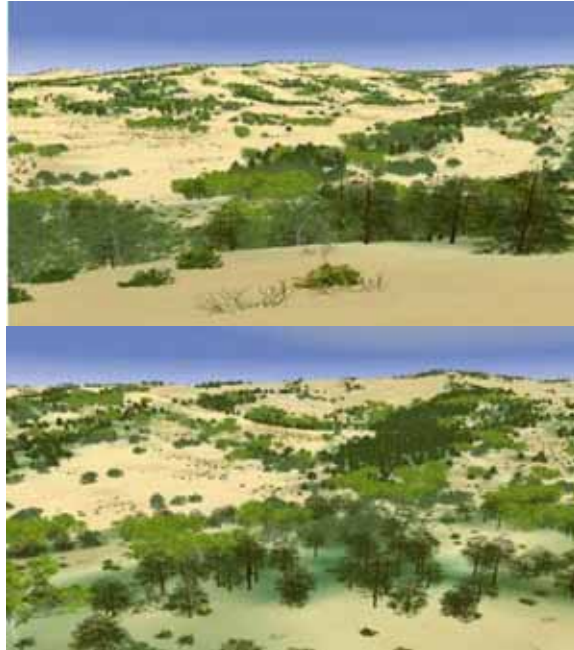


Figure 4-10 Examples of GENETICS vegetation placement with commercial objects (Bionatics® REALnat™)



Figure 4-11 Example of man-made object distribution (TIGER census data used to build road network)

G. LEVEL-OF-DETAIL AND CULLING

Once the location, orientation, and scale of a particular object model has been established, it must be added to the scene using an efficient spatial data structure such as a quadtree. As the bounding volume of a quadtree branch intersects or falls within the view frustum, that branch is considered active and potentially viewable. Non-active branches are culled away from the rendering of the current frame. Small pixel culling prevents rendering objects that do not meet a minimum screen size threshold. Finally, the

object model itself can make use of LOD or switch nodes within its own data structure to determine the appropriate representation required by the scene as a function of distance from the viewer. Thus, distant objects can be drawn as billboards, medium range objects can be represented as intersecting planes, and close objects can be depicted as full geometric objects. As noted before, some commercial packages can create LOD object models automatically and we have chosen such a package for our own work as we could not find a comparable open source solution. Our assumption remains that most simulation centers are likely to have custom-built model libraries at their disposal or have the means to quickly generate such objects as needed. We believe that creation of such objects is not the major hurdle in realistic landscape generation, but that the believable placement of millions of vegetation objects within a scene is the larger challenge.

H. AUTOMATIC GENERATION OF LCC DATA

A logical alternative to creating terrain probability distributions by hand is to use machine learning technologies to construct them automatically. We have conducted preliminary work in the use of machine learning techniques (e.g. k-nearest neighbor estimators) for automatically estimating LCC distributions. One appeal of these techniques is that the resultant formally-specified probability distribution (with its biases and assumptions) can be rigorously characterized and input back into the GENETICS system as a starting point for creating LCC regimes. Another benefit of these techniques is that they can be partially or fully automated, reducing the workload on a human modeler. The major liability of these techniques is that they require "training data" in order to function (i.e. a region in which the LCC values are known). The tacit assumption here is that the provided training data is correct and has a similar distribution to the target locale to which the technique is being applied.

Machine learning approaches to constructing probability densities require a set of training data. In our case, this training data consists of a set of exemplars that correspond to a pixel color value for which the actual LCC type is known. Each exemplar is an ordered pair consisting of domain and range values. The range value is the LCC type. The domain values can be any available quantities relevant to predicting LCC types. We

have chosen use elevation, slope, aspect angle, and relative elevation for the pixel in question. Each of these values is represented as an integer between 0 and 255 inclusive.



Figure 4-12 LCC training data shown in the upper left corner. The rest is false color elevation data.

The simplest and most successful approach we have studied so far is to estimate LCC data using a k-nearest neighbors density estimator. (Duda, Hart, & Stork, 2000) At each pixel where the LCC type is unknown (the *query point*), the domain values described above are gathered. Let us take x_i to be the vector of domain values for the i^{th} member of the training set and y_i to be the corresponding (known) value of the LCC type. Let x be the corresponding vector of values at the query point. For each query point we find the k exemplars with range values (x_i 's) closest (in terms of ordinary Euclidean distance) to the query point. The search for the nearest exemplars can be performed in log time by using data structures and search algorithms designed to make spatial range queries efficient, such as the k-d tree. (Preparata & Shamos, 1985) The probability of an LCC type existing at the query point is taken to be the percentage of k exemplars having that LCC type. Figures 4-12 and 4-13 display a region with known LCC values and the maximally-likely value of the LCC type over the entire region, including on the training set. Note that the predictions for the training region would not be used and are presented

only for purposes of comparison to the ground truth. While imperfect (some features such as urban regions are not predicted at all), this technique can help fill in gaps in LCC data.

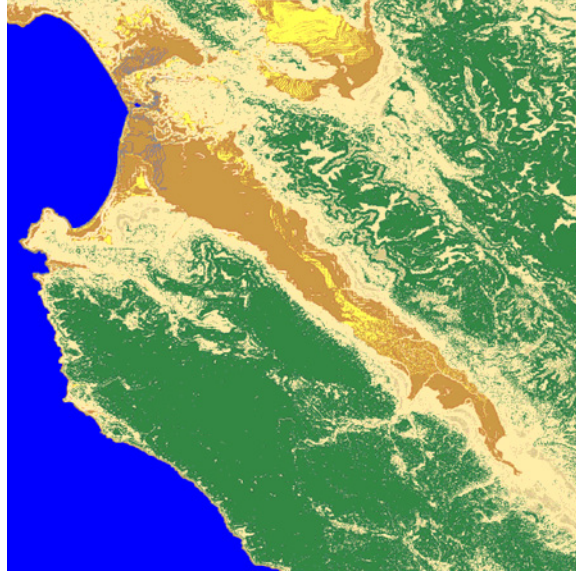


Figure 4-13 Maximum likelihood estimate of the LCC type with the k-nearest neighbors estimator.

A weakness of this model as compared to the by-hand approach is that it does not take into account the LCC type assigned to nearby pixels. If the LCC type of a given pixel is unknown, the types of neighboring pixels may be unknown as well. This difficulty may be overcome by constructing the LCC type estimates iteratively. That is, given an initial, possibly random, guess of all unknown LCC types, the LCC type at each pixel is estimated to be consistent with the guess. This estimation process is then repeated until convergence is achieved.

I. PROGRAM FLOW AND DEPENDENCIES

GENETICS' vegetation placement routines (VEGE – Vegetation Environment Generation Engine) reside within the SOARXTerrain component of the military's open source game/simulation engine, Delta3D (Darken, McDowell, & Johnson, 2005). Delta3D is an open source high-level API that sits on top of numerous other open source

libraries (see Figure 4-14) such as Open Scene Graph. (Burns & Osfield, 2003) Within GENETICS, extensive use is made of the Geospatial Data Abstraction Layer (GDAL). GDAL is an open source translator library for handling (e.g. reading, writing, manipulating) a wide variety of raster and vector geospatial data formats. GDAL allows GENETICS to integrate many disparate datasets (e.g. NGA DTED™, USGS NLCD, NASA Landsat 7 imagery, and US Census Bureau TIGER data) into one's application without the need for expensive proprietary tools.

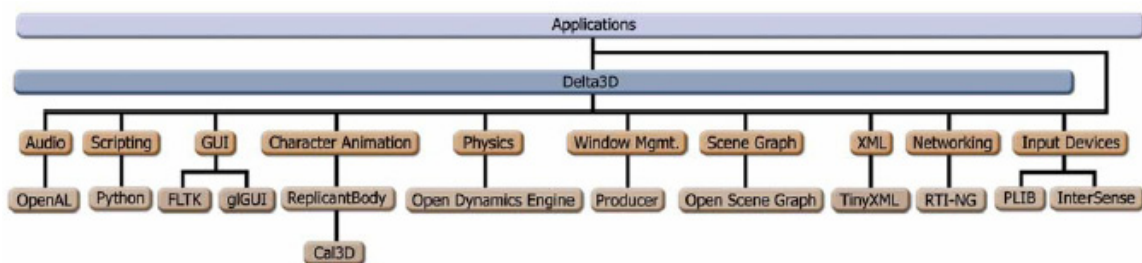


Figure 4-14 Delta3D's underlying open source libraries
(From Darken)

VEGE works in tandem with our SOARX CLOD implementation SOARXTerrain, but SOARXTerrain is not dependent on VEGE. Thus, Delta3D applications can use SOARXTerrain, but not the VEGE vegetation placement algorithm. This design allows developers to replace the terrain surface CLOD functionality with another CLOD implementation or simply load a static mesh without impacting VEGE. The primary linkage between VEGE and the rest of the SOARXTerrain class consist of the ability of the VEGE routines to read the SOARXTerrain heightfield data structure and call the GetHeight(x, y) function that returns the elevation value for a given location on the mesh which allows VEGE to perform ground clamping on the vegetation objects. In actuality, this linkage could be simplified to solely using the GetHeight(x,y) function call to create the VEGE-required topographic image maps.

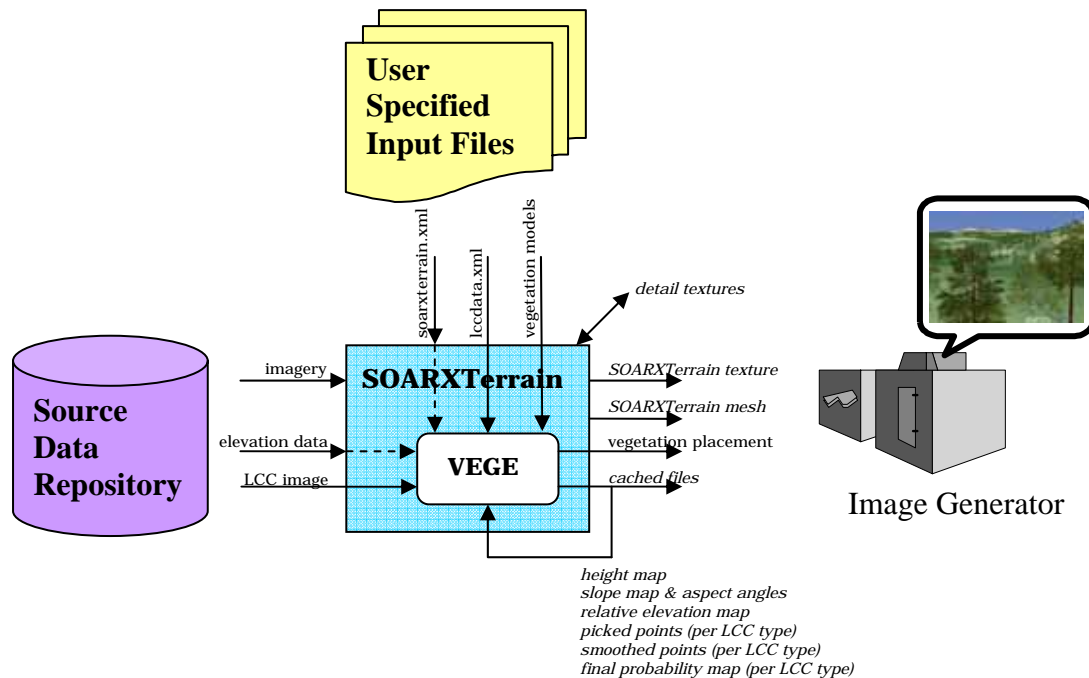


Figure 4-15 GENETICS “black box” diagram

In Figure 4-15, we depict the flow of data through GENETICS. Source data (either stored locally or accessed from a central repository), user-defined XML configuration files, and visual object models serve as inputs into the system. Output to the image generator within our simulation application consists of a scene graph containing the terrain mesh and associated textures and the vegetation objects and their placement within the scene. SOARXTerrain generates the terrain mesh, ground surface base texture, and detail textures that are cached for subsequent runs as desired. Elevation data, accessed from SOARXTerrain heightfield data, is used by VEGE to create topographic image maps. From the source LCC image, pixels are extracted corresponding to each selected LCC type and saved as a separate “picked points” image which is smoothed to allow for overlapping LCC types. Finally, a probability map for each LCC type is generated based on its smoothed image map and modified by the results of a comparison between the topographic image maps and the LCC type’s regime characteristics as specified in the LCC configuration file. All of this data is cached for subsequent runs of the same terrain. Using Open Scene Graph functionality, we can save the entire scene graph (or a portion of it) to a file during run-time, effectively archiving a

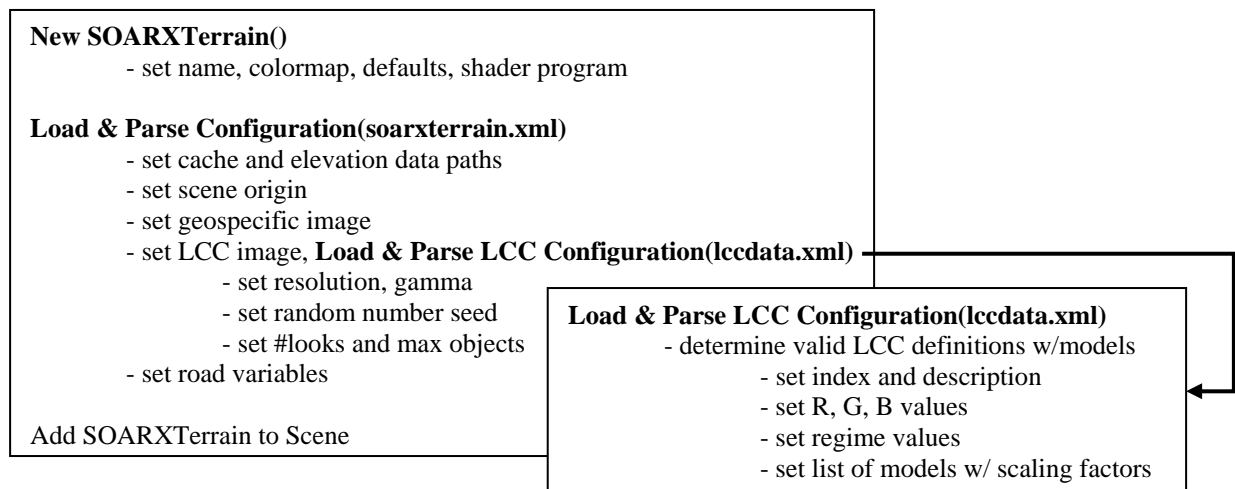
static representation of the existing scene. Thus, our system represents an implementation of the magic black box envisioned by Bitters that automatically captures raw source data and delivers a completed run-time terrain database. (Bitters, 2004)

J. ALGORITHM OVERVIEW

This section presents an outline of how the GENETICS algorithm was implemented within Delta3D's SOARXTerrain class. In the interests of readability and space, many details are left out so as not to present the reader with raw source code. Within the diagrams, bold typeface is used to identify a major functions that are outlined elsewhere.

1. Initialization and Configuration of GENETICS

In this portion of GENETICS, we initialize the system by creating the SOARXTerrain node, setting various default values, and reading the system configuration file (which may lead to parsing the LCC file). Finally, the empty SOARXTerrain node is added to the scene graph. Parsing the LCC file results in a list of valid LCC types with their matching RGB values, regimes, and models. This valid set of LCC types is referenced in later sections.



2. Simulation Application Actions That Occur Each Frame

This fragment recognizes that modules within the simulation application can have an effect on the terrain visualization. As an example, keyboard input is used to modify various settings within SOARXTerrain.

GetKeyboard

- set Threshold, Detail, Time of Day
- set xyz, hpr
- set movement model
- display position, statistics, vegetation

3. SOARXTerrain Class Actions That Occur Each Frame

The following outline looks processor-intensive and it can be since it holds the potential of calling not only the LoadSegment method, but its related counterpart AddVegetation. Note that these methods are only executed fully when no terrain mesh (or vegetation subgraph in the case of AddVegetation) exists for the desired cell. When LoadSegment is called, it first checks to see if a Segment exists for this cell and exits the method if true. Otherwise, a Segment is created and elevation data is loaded up into a heightfield and the textures needed to cover the terrain mesh are loaded or produced. If we are interested in placing vegetation on the surface, we load or generate topographically-derived images for the cell (heightmap, slope/aspect map, relative elevation map) and for each valid LCC type, we load or generate their set of images (picked points, smoothed map, and final probability map) from the source data as described previously. Finally, the AddVegetation method (described in the next section) is called to create the vegetation subgraph.

```

- get eye point
- get current latitude/longitude, load distance, lat/long extents of viewer
- while (within extents)
    LoadSegment(latitude, longitude)
        - check if Segment is already loaded and return if true
        - find elevation source data
            - check for existing mesh data else create mesh
            - load elevation into heightfield
            - check for existing detail and scale gradients else create
            - set detail and scale gradient images into textures
            - check for existing base color and gradient else create base
              color from imagery/elevation & gradient from heightfield
            - set base color and gradient images into textures
        - if using LCC
            - check for existing LCC color image else create
            - for all valid LCC types
                - check filter image & load else create filter image
                - check smooth image & load
                  else create smooth image (w/ masking as req'd)
            - check for existing heightmap
              else create from heightfield
            - check for existing slope and aspect map
              else create s/a maps from heightfield
            - check for existing relative elevation map
              else create relative elevation map from heightfield
            - for all valid LCC types
                - check for existing probability map
                  else create probability map using smoothed image,
                    height map, slope map, & rel elev map
        - set origin, transformation node & attach shader program
        - if using roads
            - check for road geometry else create roads
        - add Segment to scene graph

- if using LCC
    - AddVegetation(latitude, longitude) → ○

```

4. Expansion of the AddVegetation() Method

This outline is where each LCC's probability map is matched up to a geometric model and placed within the scene. A full quadtree is created for the cell and then pared back based on the nonexistence of vegetation objects to help minimize culling traversals. Collision detection between vegetation objects is accomplished by comparing the bounding volume of a candidate object against the bounding volumes of all sibling objects within the same left group. If a significant overlap occurs, the candidate object's placement is rejected. Shrub placement is exempt from the collision constraint, while

collision between urban objects is more rigidly enforced. Note that our chosen random number seed is reinitialized for each geocell, guaranteeing the same results (with the same configuration parameters) even if players start in different cells. Not shown in this outline is the reuse of model nodes by linking each position-attitude-transformation node to a single shared model which also contains LOD nodes (see Figure 4-16).

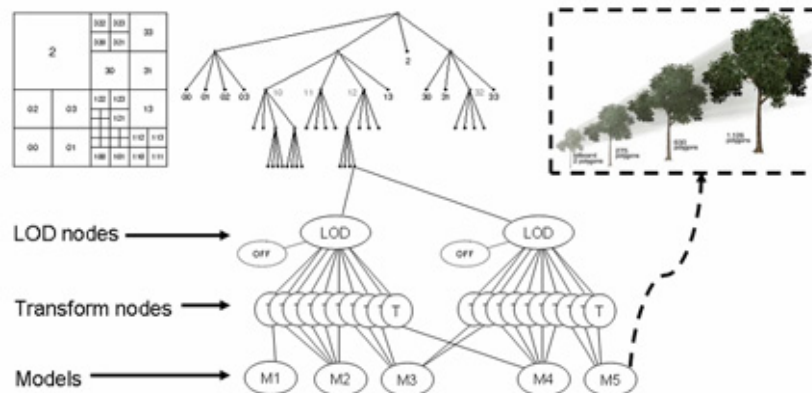
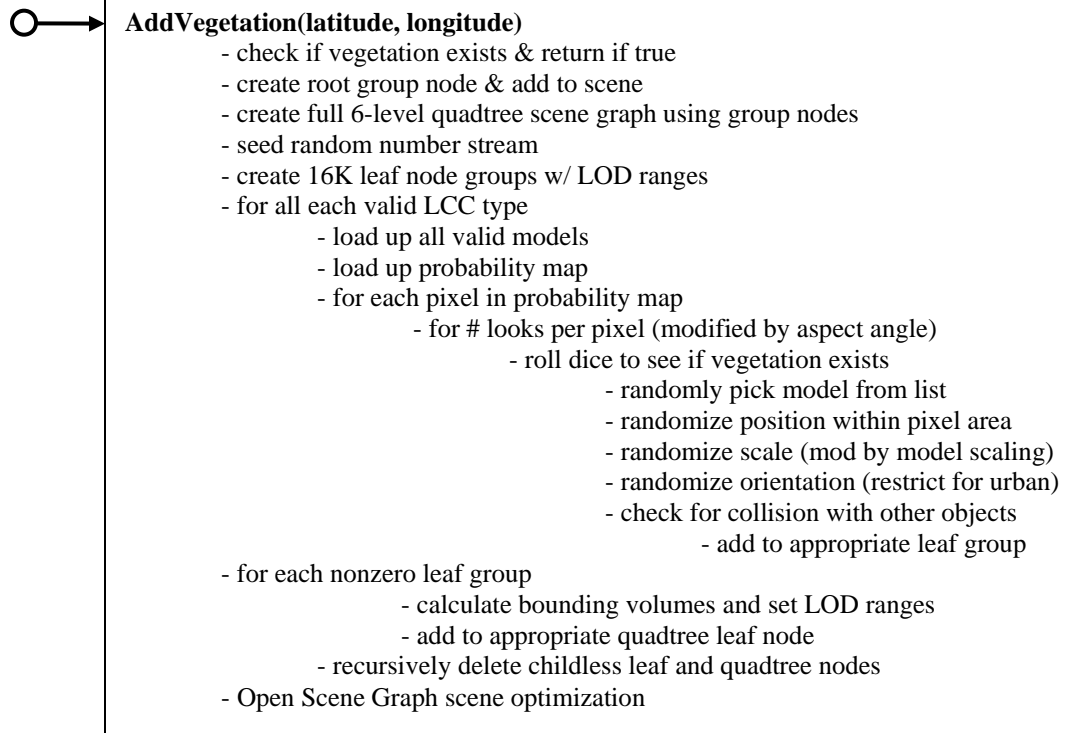


Figure 4-16 GENETICS quadtree structure

V. RESULTS

A. OVERVIEW

This chapter addresses the results obtained from testing the GENETICS system across multiple platforms in a variety of conditions. We looked at the startup and rendering performance of GENETICS, the ability to consistently recreate the same natural environment across multiple machines, the ability to vary the environment by changing the random number seed, visually comparing the geospecific accuracy of GENETICS, comparing GENETICS placement using raster and vector land cover data, and finally addressing the issue of cost and timeliness to create terrains. As part of the consistency test, we also looked at the addition of networked players to the environment. The following sections discuss each of these tests in greater detail.

B. PERFORMANCE

In this section, we describe our three machine configurations and relate both the initial processing and runtime performance from multiple GENETICS runs. Before we present the results, we outline the configurations of the three test machines.

The first GENETICS test machine (“Genetics”) is a desktop system with an AMD Athlon64 FX-55 CPU and 2Gb RAM, a NVIDIA 6800GT 256Mb graphics card, and a Seagate Barracuda 120Gb hard drive operating at 7200 rpm. The second GENETICS test machine (“Gargoyle”) is a desktop system with an AMD AthlonXP 3000 CPU and 1Gb RAM, a NVIDIA 6800GT 256Mb graphics card, and a Seagate Barracuda 80Gb hard drive operating at 7200 rpm. The third GENETICS test machine (“Voodoo”) is a laptop with an AMD Athlon64 3000 CPU and 1Gb RAM, an ATI Radeon Mobility 9600 64Mb graphics processor, and a Hitachi Travelstar 60Gb hard drive operating at 7200 rpm. The laptop ran on AC power with all power-saving features disabled.

In the Table 5-1, low resolution represents creation of a 1024x1024 pixel probability map while high resolution represents a 4096x4096 pixel probability map. The number of looks is the number of times that each pixel within an LCC’s probability map is examined (i.e. each look represents a single “dice roll” against the pixel’s

probability value). Thus, the number of looks represents the maximum number of possible objects of a particular type within that pixel's physical area. The number of objects created within a geocell is a function of the resolution of the probability map, the number of looks per pixel, the number of LCC probability maps created, the values of the pixels within the probability map, and the random number stream. Collisions between placed objects (and any subsequent deletions) also factor in to reduce the total number of objects within a geocell. The resolution, number of LCC types, number of looks, and the configuration of the machine all affect the startup time of the application. The total number of objects and the system configuration affect the rendering speed of the application in various scenes, but extensive use of quadtrees, small feature culling, and LOD nodes localize performance problems to regions of dense vegetation. For all performance testing, the viewport was configured for a 1024x768 window with a 60 degree horizontal and 45 degree vertical field of view. Networking and player rendering were disabled. Small feature culling was set to 10 (i.e. objects smaller than 10 rendered pixels were culled).

Table 5-1 GENETICS Runtime Performance

System	Res.	#LCC	#looks	#objects	w/o cache	w/ cache	Scene 1	Scene 2	Scene 3
Genetics	Low	4	5	537900	0'46"	0'28"	68.6	47.7	35.0
	Low	4	10	922500	1'20"	0'45"	43.6	32.8	23.2
	Low	4	20	1800696	2'08"	1'35"	24.1	19.6	13.3
	High	4	1	2071294	*	1'58"	20.8	17.2	11.6
Gargoyle	Low	4	5	537900	1'04"	0'29"	44.0	32.8	23.1
	Low	4	10	922502	1'17"	0'45"	28.2	21.8	15.0
	Low	4	20	1800697	2'34"	2'02'	15.1	12.2	8.5
Voodoo	Low	4	5	537900	1'01"	0'25"	30**	24**	24.0
	Low	4	10	922500	1'16"	0'43'	29**	21.0	16.9
	Low	4	20	1800696	2'28"	1'48"	15.6	11.8	9.8

* required use of a third party memory manager that skewed execution time results.

** laptop refresh limits prevented gathering of accurate results – number reflects lower bound

Table 5-2 GENETICS Viewport Objects

Res.	#LCC	#looks	#objects	Scene 1	Scene 2	Scene 3
Low	4	5	537900	1472	1680	2592
Low	4	10	922500	2472	2904	4497
Low	4	20	1800696	5088	5768	9096
High	4	1	2071294	9419	10122	18445

While rendering speed is not a major concern of our research, we have included it here for the sake of completeness and for future research in large-scale vegetation rendering. Table 5-2 reveals the number of objects rendered in each scene. Note that these figures do not reflect all the possible objects within our view frustum due to small pixel culling and level of detailing by both individual objects and groupings of objects. Also note the tiny ratio of rendered objects versus the total number of objects in the geocell. The difference between these two numbers reveals the number of objects culled from the scene. The frames per second scores are provided by Delta3D's statistics counter and represent the steady-state condition of a non-moving viewer. It should be noted that geometric object instancing (Carucci, 2005), still a graphics library dependent feature, holds the promise of dramatically increasing frame rate by storing an object's vertex and texture data on the graphics card and directly passing position, attitude, and transform information to a shader program vs. storing this information as static nodes within the scene graph. At this point in time, OpenGL does not support this functionality. Additionally, we have maximized the use of textures as a storage medium to facilitate the expanded use of shader programming to improve the performance of future versions of GENETICS.

It is interesting to note that the AthlonXP machine produced slightly different results in the total number of created objects than the Athlon64 machines (2 additional trees in the 10 looks case and 1 additional tree in the 20 looks case). From run to run, the total number of objects did not change, suggesting that the machine created consistent random number streams. Indeed, when collision detection between vegetation objects was turned off, all three machines reported identical figures. Within the collision

detection algorithm, a comparison is made between the distance between the position of two objects (an already placed object and a potential new object) and one fourth the radius of the bounding sphere of the potential new object. It is likely we are witnessing a precision error that may be attributable to Athlon64 processors running in a compatibility mode under 32-bit operating systems such as Windows XP Pro.

C. CONSISTENCY AND VARIANCE

In this section, we discuss the testing conducted to ensure consistency of vegetation distributions within the environment. We looked at three GENETICS-produced scenes (i.e. specific position and orientation of the viewer). Each scene was created three times, both with and without cached data. Screen captures of the scenes were recorded and visually compared for differences. Additionally, we recorded the position, scaling, and orientation (i.e. rotation angle) for objects located within a given cell in the environment. This battery of tests was conducted for three different random number seeds on each of the three test machines. We chose the same settings as one of our performance test cases: 4 LCC types at low resolution with 10 looks per cell. In order to remove any issues concerning the collision detection anomalies noted above, we disabled collision detection during vegetation object placement.

Each of the machines produced the same positions, orientations, and scaling factors for the objects located within our test cell when using the same random number seed. This was true for each of our three seeds. No variation occurred between cached and non-cached runs using the same random number seed. Likewise, each of the three scenes was perfectly replicated when using the same random number seed. A sampling of the consistency/variation test results are presented in Appendix B. An example of how vegetation varies when using different random number seeds is shown in Figure 5-1. Note in this figure how overall landmark features of the terrain are preserved (e.g. forest shapes and composition) while localized details change (location and/or existence of individual vegetation objects).



Figure 5-1 Scene 1 with random number seeds 42, 47, and 12.

1. Networked Players

As a further examination of consistency, two machines (“Genetics” and “Gargoyle”) were each configured with a networked player (helicopter and tank, respectively). These players were linked using the Runtime Infrastructure (RTI) services of the High Level Architecture (HLA) and placed in several scenes to compare their view of each other in relation to their shared synthetic natural environment. Once again, we used the same settings and random number seeds to guarantee consistency.



Figure 5-2 GENETICS networked player test

In Figure 5-2, we see an example scene from the networking test. In the first picture, the tank player (“Gargoyle”) is facing a sugar maple tree (i.e. light green leaves with an off-white trunk). There are three bushes to the left of that tree and another sugar maple to the right. A tall pine tree can be seen beyond the shrubs. The tank’s turret is

pointing to the right of the screen. In the second picture, the helicopter player (“Genetics”) is positioned behind, above, and slightly to the left of the tank. We can see from the helicopter’s vantage point that the vegetation seen from the tank player’s view is positioned identically within the helicopter player’s application. Note that the tank player’s system clock was several hours ahead of the helicopter player’s, resulting in a darker ground texture.

The networked players test demonstrated that not only could GENETICS create a consistent representation of the natural environment on different machines, but that by doing so, it became extremely difficult to locate another player within the environment. For further examples, see Appendix C.

D. ACCURACY

In this section, we examine testing conducted to gain a sense of where along the geotypical/geospecific continuum the GENETICS algorithm falls. For this series of tests, distinctive regions of terrain were chosen and screenshots were taken of the test condition and compared against each other distributions and “ground truth” imagery. We generated distributions for four different LCC types (shrubs, deciduous, evergreen, and mixed forest) and in high and low resolutions. A Hammes-like test was infeasible as all the test LCC types belonged to the same Hammes ecotope (i.e. “dense brush”). Comparing GENETICS to a vector-derived vegetation placement scheme using a commercial tool was also infeasible due to an expired license that prevented the creation of a terrain database. While this was a temporary issue, remedied with the expenditure of research funds, it underscores a fundamental limitation in using license-based proprietary software. Instead, a rasterized version of a high resolution LULC vector dataset was created and tested as described in the “Raster vs. Vector Data” test section.

It should be noted that we do not know the pedigree or projection of the imagery used in our “ground truth” representations (courtesy of Google Earth). Secondary “ground truth” applications (e.g. TerraServerUSA, NASA’s World Wind) revealed that our test regions contained a composite of 1993 and 1998 high resolution aerial photos.

For the Google Maps and Earth images, it appears this high resolution aerial photography was mixed with more recent, but comparatively low resolution, Landsat 7 14.5 meter color data (i.e. the imagery used for our ground surface texture).



Figure 5-3 Google Maps imagery vs. GENETICS vegetation distribution of triangular vegetation growth near Watsonville, CA

For our first test region (a triangular patch of vegetation growth near Watsonville, CA, see Figure 5-3), we compared the featureless terrain representation of NASA's World Wind software versus the vegetation-laden representation from a GENETICS-based simulation (see Figure 5-4). Note the incorrect positioning by World Wind of the Landsat 7 imagery (lower left) that is correctly positioned in its GENETICS counterpart (lower right). Road data comes from U.S. Census Bureau's TIGER dataset. Road network visualization was not used in subsequent tests since it is not directly a part of our research. The high resolution 1:24K topographic map is provided to show the minimal amount of vegetation detail available to a vector map product. Note that the vegetation distribution seems to fall close to the "actual" distribution as seen in the high resolution imagery. Trees are missing from along the major road following Corncob Canyon. Trees are present in similar density and placement along the gently-sloped canyon walls. The overhead shot in Figure 5-3 shows how the overall effect of the vegetation placement preserves the landmark features of this terrain from an aerial perspective. Note also that differing projections are being used to display the images in Figure 5-3 which can account for some of the discrepancies found in the Figure 5-4.

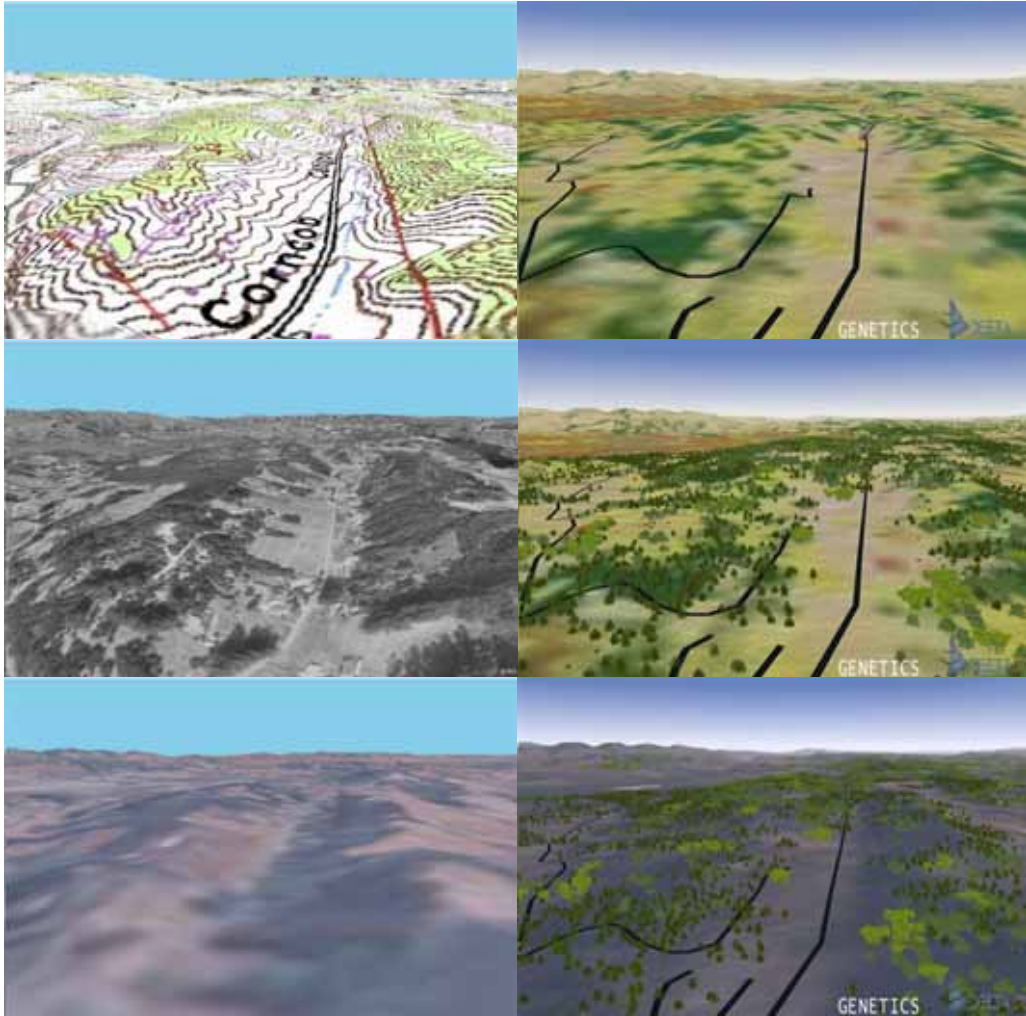


Figure 5-4 NASA's World Wind (left) vs. GENETICS (right)

After some initial success in recreating the triangular vegetation growth scene, we looked at a more complicated series of tests. We generated vegetation distributions for an equally distinctive geospecific landmark, a rectangular eucalyptus grove near Aromas, CA (see Figure 5-5). As a point of reference, this grove is nearly 1km wide. Note in Figure 5-5 two small white lines that appear just below the center of the image. The smaller white line approximates the smallest pixel width of our “high resolution” LCC dataset (i.e. 25m) while the larger white line represents our “low resolution” LCC dataset (i.e. 100m). Thus, a high resolution LCC pixel notionally represents a 25m x 25m area while a low resolution LCC pixel would represent an 100m x 100m area.



**Figure 5-5 Google Earth image of Aromas, CA eucalyptus grove
(note 25m and 100m lines)**

In this second series of tests, we looked at a variety of vegetation distributions: random, random with topographic modifications, strict LCC placement, and a GENETICS placement (i.e. LCC placement with topographic modifications). For the random distribution, a noise texture replaced the smoothed picked points texture. Using a noise texture is actually more realistic than a simple Poisson distribution as probabilities smoothly transition between high and low values, resulting in variation of vegetation density versus an even distribution. Once again, we compare these distributions against the actual imagery of the same area. Both high and low resolution tests were conducted with the same four LCC types using 1 look per pixel and 20 looks per pixel respectively.

As expected, randomized distributions did little to convey any real sense of place (i.e. geospecificity), but would have added plenty of cover for ground targets. Adding topographic influences to the random distribution helped to define major topographic features (see Figures 5-6 and 5-7). Figure 5-6 shows the high resolution, purely random distribution. Even when the topographic influences are accounted for, there is no ability to recognize distinctive landmarks and obviously, the eucalyptus grove is absent. Figure 5-7 shows a low resolution, purely random distribution where no distinctive terrain landmarks are present and the relative sparseness of the vegetation works against the ground texture, confusing the viewer as to what is actually being represented.

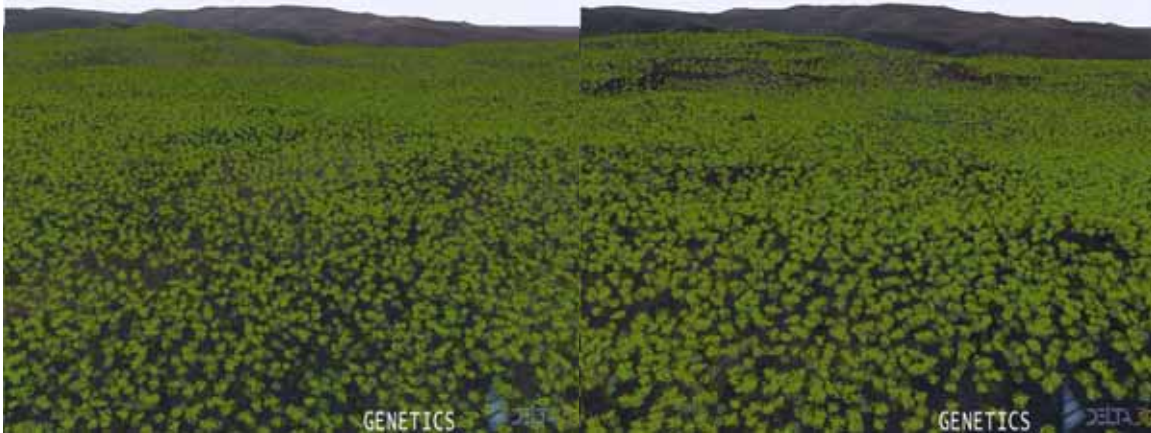


Figure 5-6 High random distribution with and without topographic influences

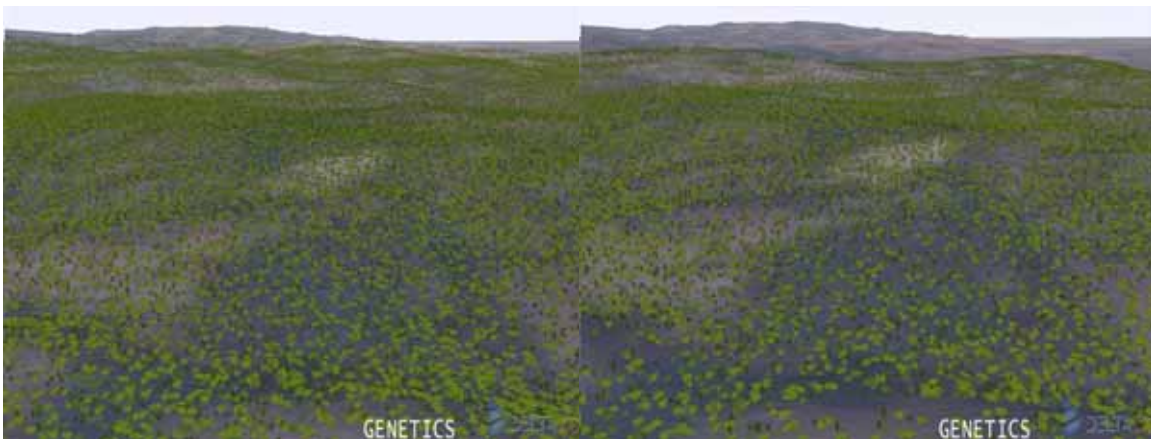


Figure 5-7 Low random distribution with and without topographic influences

In Figures 5-8 and 5-9, high and low resolution GENETICS-generated vegetation is placed within the environment and compared against imagery of the terrain. Even with the slightly differing orientations of the images, the rectangular shape of the vegetation is recognizable and matches well to the underlying imagery. Additionally, in Figure 5-8, we show the ability to vary the density of the vegetation by changing the number of looks each probability pixel receives. Note that the relatively sparse low resolution GENETICS distribution gives a good approximation of the dense high resolution case with significantly less processing required by the system.

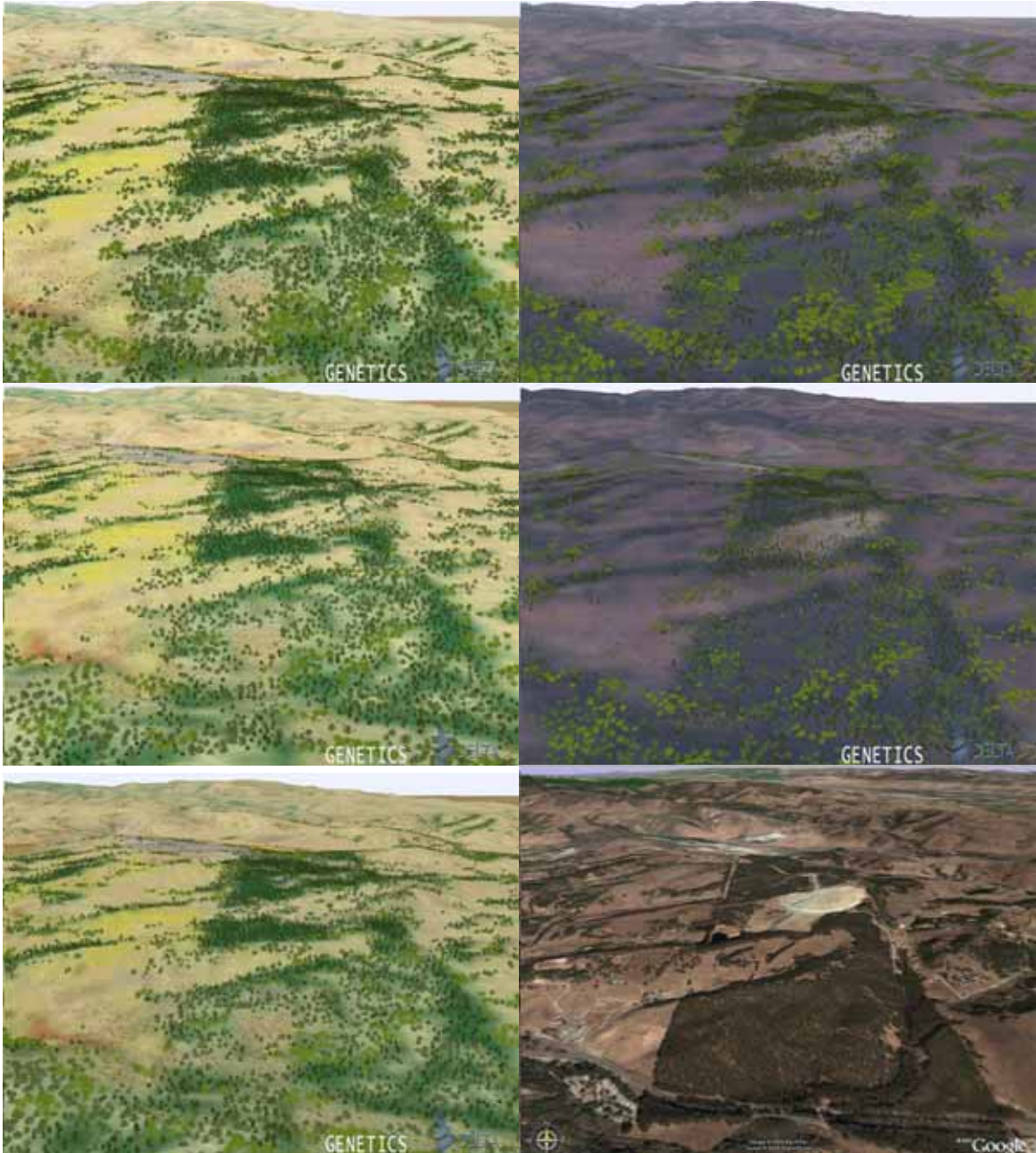


Figure 5-8 Top: GENETICS high with 2 looks per pixel; Middle: GENETICS high with 1 look per pixel; Bottom: strict high resolution LCC placement and imagery

We also conduct a test using just LCC data without any smoothing or topographical influences. Strict land cover placement in high resolution mode provides a good correlation to the imagery despite the lack of density variations due to topographic features (see Figure 5-8: bottom left). In the low resolution case, the smoothing and

topographic routines are needed to keep the vegetation from looking like a patchwork quilt of distinctive vegetation zones (see Figure 5-9, lower left).

Both the high and low resolution GENETICS-based placement demonstrated nearly the same level of correlation to the imagery as the strict LCC placement, but also contained variations in density due to topographic and proximal features within the environment. The GENETICS scheme also showed a smaller degradation in correlation between high and low resolution modes.

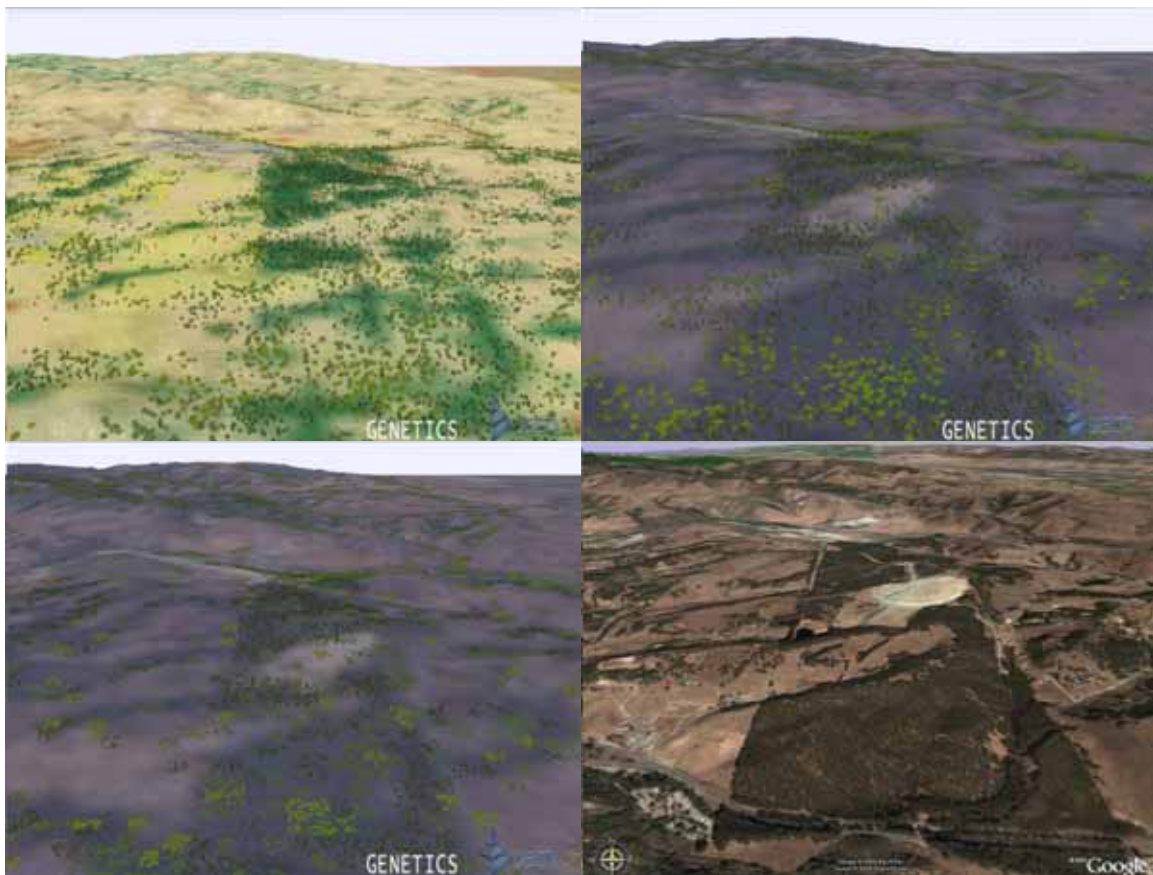


Figure 5-9 Top: GENETICS low with 20 looks per pixel; Bottom: strict low resolution LCC placement and imagery

E. RASTER VS. VECTOR DATA

In this test, we compare differences in geospecificity with regards to using raster and vector datasets. For the raster dataset, we use our standard NLCD 1992 GeoTIFF

source images. For our vector dataset, we take high resolution (1:100K) LULC data and export a rasterized version with the commercial tool, Global Mapper®. This rasterized LULC dataset is colored in accordance with NLCD color values. Note in Figure 5-10 the coarseness of the LULC data with respect to the NLCD data of the same area. For example, the prominent rectangular eucalyptus grove has been turned into a large mixed forest without clearings or evidence of the mining operations present (see Figure 5-11).

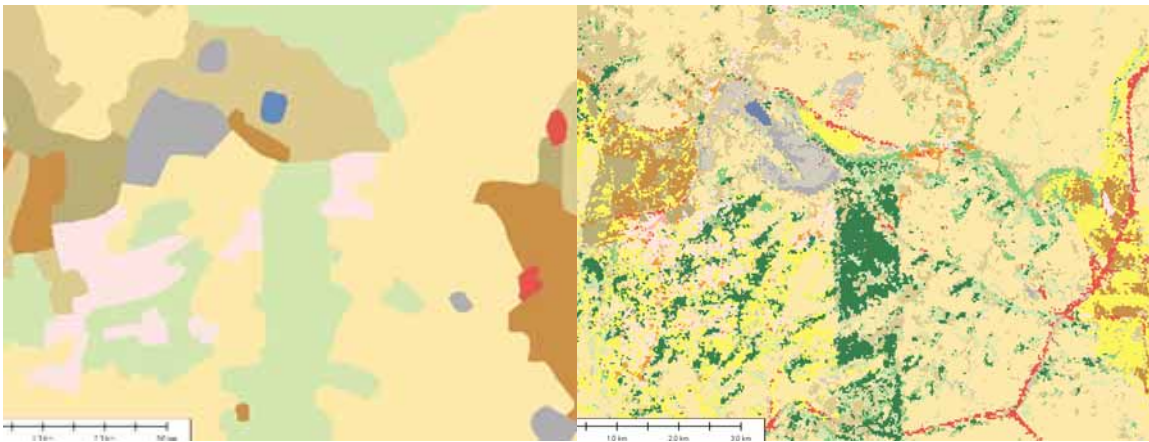


Figure 5-10 Comparison of LULC data (left) vs. NLCD data (right)



Figure 5-11 Aerial imagery of eucalyptus grove

Given the disparity of the input data, it should come as no surprise that the difference in the resulting GENETICS vegetation placement using these two datasets is likewise stark. In Figure 5-12, we see from the top images how the vegetation is placed

in regards to the underlying LCC data (as previously shown in Figure 5-10). In the bottom images, we see how poorly the LULC distribution matches to the satellite imagery as compared to the NLCD distribution.

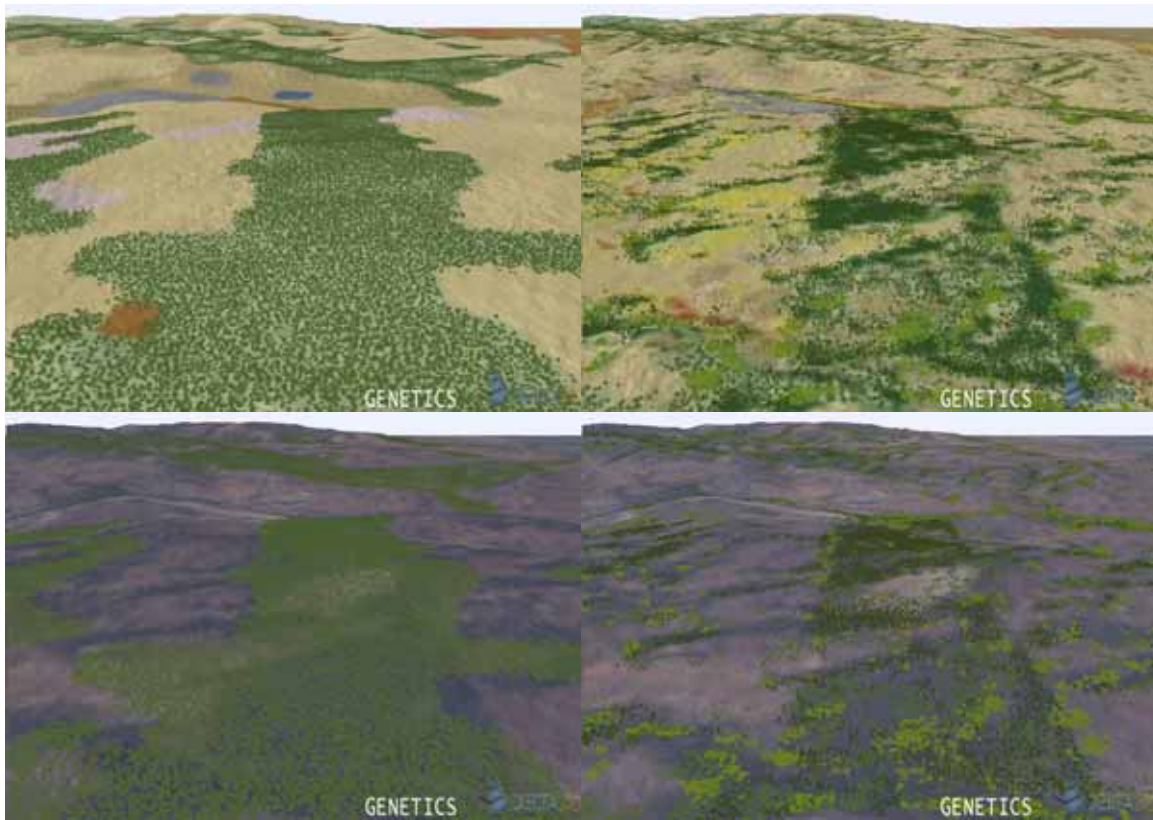


Figure 5-12 Comparison of LULC (left) vs. NLCD (right) vegetation distributions with LCC-based (top) and imagery-based (bottom) ground textures

The results of this test show that “high-resolution” (i.e. 1:100K) LULC vector data is no match for 30m NLCD raster data. The relatively poor resolution of the vector data results in large blob-shaped regions that are not able to accurately portray detailed landscape boundaries. This limitation also forces vector data to sacrifice fidelity by grouping multiple LCC types into a single heterogeneous coverage. In the example above, the NLCD data is able to capture evergreen, deciduous, and mixed forest types within the eucalyptus grove while LULC aggregates these distinctions into a single “mixed forest” classification. Finally, it should be noted that while LULC data fared poorly against NLCD, both datasets (once LULC was converted to a GeoTIFF format)

were able to be used by GENETICS to automatically place vegetation objects. This capability alone is a benefit over traditional vector-based terrain modeling systems since GENETICS was able to soften the boundaries between LCC zones by smoothing the vector data and accounting for topographic influences.

F. COST AND TIMELINESS

GENETICS is freely downloadable as a component of Delta3D and online tutorials and technical support are available through the Delta3D website (i.e. www.delta3d.org). The academic cost of our in-house commercial terrain database tool is \$35K, plus \$6K yearly maintenance fee, plus \$5K for training. A user of a terrain database generated with our in-house commercial tool will need to purchase this tool and be instructed in its use in order to modify or update their terrain. Depending on the output format of the terrain database, the terrain may only be modifiable by the particular proprietary tool that created it.

Our resident artist claims he can reproduce a similarly detailed terrain database of a geocell using commercial tools in about a week. GENETICS can create this geocell-size environment in a matter of minutes and if the player travels beyond the boundary of the initial geocell, creation of the new geocell takes a few more minutes. Spawning the creation process prior to reaching a boundary provides for a seamless transition between cells. By automatically creating terrains directly from raw source data, GENETICS allows users the freedom to adopt a philosophy of always using the “latest and greatest” source data from their repository without needing to walk new data through the terrain database modeling process.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. SUMMARY, CONCLUSION, AND RESEARCH AGENDA

A. SUMMARY

This dissertation was borne out of recognition of critical limitations in the visual fidelity of current combat simulators with regards to terrain. Detailed landscapes are largely missing within these training systems and yet computer graphics techniques developed over the past 20 years have hinted at or described in depth various ways to fill in portions of this missing detail. Many of these techniques have never been looked at for use within a networked combat simulator, much less automating the process of terrain database generation from a minimum of source data. Our proposed system, GENETICS, is designed to be common across a wide spectrum of military simulators. Dismounted soldiers, tank drivers, and helicopter pilots can use the GENETICS terrain generation and visualization system within their respective simulators, to display the same perceived terrain at the level of detail required by that simulator to meet its performance constraints.

This dissertation has laid out the framework for such a system. An efficient continuous level of detail algorithm, SOARX, was implemented within our open source simulation/game engine, Delta3D. Our SOARX implementation automatically loads up raw elevation data (e.g. DTED™) that is processed into optimized, view-dependent geometric meshes as the viewer approaches unvisited regions. Noise detail is added between known elevation postings to increase the apparent quality of the data. These noise-generated features are reflected within matching detail textures that are combined with the region's base texture (e.g. satellite imagery) and corresponding normal map to create appropriate terrain shading within a shading language program at runtime. Fog and atmospheric lighting are included as functionality within Delta3D's weather and environment classes.

The major issue that this dissertation addresses is the realistic placement of vegetation objects within a large scale environment without the need for proprietary tools or skilled terrain modelers. By using medium-resolution (e.g. 30m postings) land cover classification data as the basis for such placement, we achieve an improvement over geotypical placement techniques without such a grounding to real-world data. Noting

that raster data is unable to provide overlapping ecotypes, we first separate each ecotype into its own dataset and then smooth the data based on proximal relationships to other matching ecotypes. While this configuration now allows for overlapping coverage, the density of each ecotype within a particular cell is still largely unknown. Based on previous work in modeling ecosystems, we take a range of elevation-derived products (elevation, slope, aspect angle, relative elevation) to modify the smoothed datasets to create probability maps. By tweaking ecotype parameters, we can adjust the likelihood of vegetation objects to exist within a particular regime. If a specific ecotype is able to withstand steep slopes, the negative effects of an object being located on a steep slope are lessened. To add in variability within an ecotype, GENETICS allows the use of multiple geometric models with associated scaling factors, and random offsets to scaling, position, and orientation. Stable interactive rendering speeds remain a challenge and currently limit the number of objects that can be placed within the terrain. It is hoped that future work in vegetation rendering and shading language programming will dramatically increase the number of objects within a particular scene.

Finally, this dissertation offers a look ahead at a new philosophy for creating virtual terrains. Traditionally, the simulation community has relied on skilled terrain modelers to create custom-tailored executable databases from a master database using expensive proprietary tools. This process is time consuming and inefficient since the modeler must compile a customized product for each simulator. In order for a simulator to make use of updated source data, the master database must be first updated and then executable versions are created and sent to each simulator. Each simulator must be updated at the same time in order for the terrains to remain consistent with each other. GENETICS takes the approach that simulators should be responsible for quickly creating their own virtual terrains from a common repository using the latest source data. If there is a need to guarantee consistency (e.g. a networked simulation), parameters are shared between players to generate the same set of terrain objects in the same manner. Given the same image generator, these environments will look identical to one another. No longer is a simulationist held hostage to the availability of skilled artists and their tools. Simulationists can create specialized versions of a terrain or multiple versions of the

same terrain without needing to task a modeler. The time required to create these terrain is now measured in minutes and seconds versus days, weeks and months. Finally, this automated process frees the terrain modeler from the tedious task of determining where vast amounts of vegetation objects are placed to instead focus on building geospecific features such as urban environments which can be incorporated into GENETICS-generated landscapes.

B. CONCLUSION

We have proved that GENETICS is able to procedurally create identical vegetation object distributions between networked simulators. The burden of creating large-scale natural environments is now automated into a task that can be accomplished during run-time execution of a simulation application. Consistency is guaranteed by sharing common source data and a small collection of application settings. Once the probability maps are generated for a particular region, the change of the random number seed will alter the location and orientation of all of the vegetation objects while maintaining the overall look and feel of the terrain. This helps trainers overcome players learning/memorizing the details of a particular static scenario versus requiring the players to respond to whatever environment they are placed within. Players are now forced to practice terrain-based situational awareness (SA) and tactics in order to survive.

GENETICS provides a host of improvements over typical terrain databases and terrain database generation tools. GENETICS offers variable vegetation density with overlapping vegetation types. Terrain object placement is responsive to its environment; allowing vegetation to react to both landscape contour variations and neighboring landcover types. A modeler's time is not wasted on tediously placing millions of vegetation objects or building simulation platform-specific compilations of static playboxes. By providing a uniform set of object types, positions, scalings, and orientations to the application's image generator, GENETICS allows these highly optimized hardware and software components to determine the best way to visualize the scene without needing to create unique terrain databases for each image generator. The onus of creating the best possible scene is now placed where it should be, on the image

generator, versus the input data provided to the application. As new graphics hardware becomes available, it can be swapped into the simulation system without needing to update any terrain database.

GENETICS also represents a new paradigm in terrain generation and visualization. Visualizing synthetic natural environments no longer requires a modeler to build and texture a terrain mesh, assemble a collection of appropriate objects, and then properly place them within the environment. Automation and a set of production rules allows one to realize substantial temporal and financial savings in generating virtual environments. This situation gives users unprecedented flexibility to exploit continually-updated datasets. Each time a representation of the terrain is created, GENETICS can use the latest available source data. This capability is not only invaluable to a military commander responding to a crisis, but also to any decision-maker with the need to immediately visualize current geospatial data in three dimensions. GENETICS is a real-time 3D geospatial visualization tool that can also be applied to traditional GIS applications. Landcover classification data and topographic influences can be replaced with population profiles and household incomes, crime statistics, education levels, etc. Objects or icons placed within the environment can be stratified in size and/or color to represent comparative worth. Decision-makers can now immersively interact with their geospatial datasets looking for patterns and confluences that might indicate an emerging trend. Turning our attention back to military simulations, GENETICS is not simply limited to platform-based simulation applications. While GENETICS produces a scene graph for image generators, this list of object information (i.e. type, position, orientation, scale) can be easily exported to a text file for use in creating plan-view representations of the environment (e.g. maps) for constructive simulations. This one-to-one correlation of the natural environment between virtual and constructive level simulations is currently unavailable. GENETICS represents a major step forward in remedying that situation.

C. FUTURE RESEARCH

A listing of follow-on tasks within the broader scope of the GENETICS research effort is given in Appendix D. What follows is a descriptive outline of some of the

analysis, implementation, and testing techniques that can be employed during future research with respect to demonstrating improved simulator-based training using GENETICS-based environments.

Once GENETICS has been optimized, productized, and fielded, as simulationists we must ask ourselves, “Do the visual improvements offered by GENETICS actually improve training?” To answer this question, we must examine a particular training scenario within an existing simulator, identify missing terrain elements and visual cues within the scene that are normally found in the real world, ask subject matter experts (SME) on the importance of those visual elements to accomplish mission-essential tasks, and then compare the visual quality of our GENETICS system to the existing system. Consider the following example:

Mission-Essential Task #7:	Emergency helicopter landing
Training Requirement #4:	Estimate wind speed and direction
Supporting Visual Effect:	Movement of trees
Importance of visual element (0-5):	4
Exists within current simulation system:	No
Exists within GENETICS-based system:	Yes*

* note that this capability may require enhanced tree models or third-party tools (e.g. REALnat, SpeedTree)

Here we have identified a particular task and training requirement, recognized a visual effect that could support that task, asked our SME to quantify the importance of that visual effect to accomplishing that training requirement, and then noted whether such effects can be found within the current system or a GENETICS-based system. While such results are subjective due to SME input, it does begin to address visual fidelity requirements needed to meet certain simulator-based training requirements. Training requirement/visual fidelity matrices don’t exist and thus simulation creation remains a spiral development process (i.e. developer demos work to sponsor, sponsor critiques product, developer goes back to correct/improve code and demos work again to sponsor who critiques...) until the sponsor determines the simulator looks “good enough” or development stops due to some unrelated external constraint (e.g. time and/or funding runs out). Clearly, if an authoritative training requirement/visual fidelity matrix did exist, both the sponsor and developer would know what level of visual fidelity is needed to

make a simulator “good enough” for fielding and the developer could better determine the level of effort (e.g. time and money) needed to meet the sponsor’s requirements.

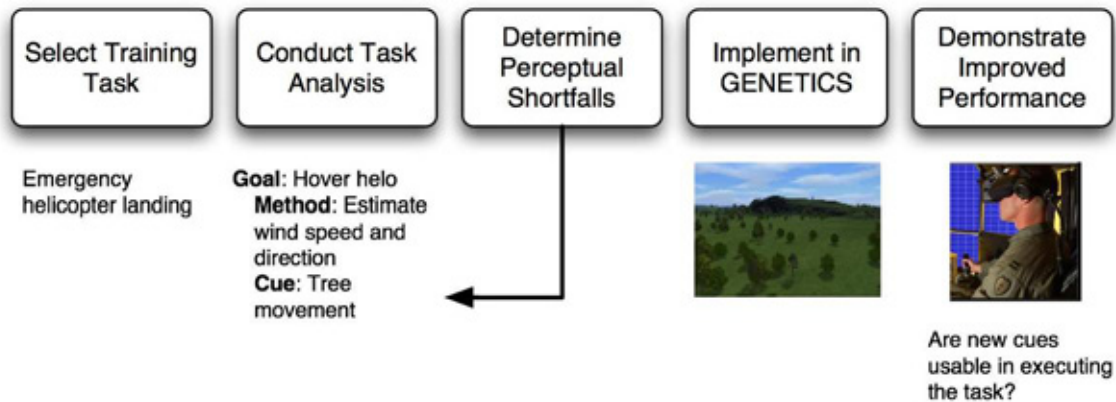


Figure 6-1 Training requirement/visual fidelity matrix research process

So why wait to do this analysis at the back-end of the process when it makes more sense to pinpoint visual fidelity shortfalls up-front and then apply perceptual improvements to those simulation shortfalls and thus improve training? This is exactly what we propose for future research (see Figure 6-1). First, focus on a specific platform simulator and a set of training tasks. Through instructor interviews and a review of instructional materials, extract the goals and underlying methods used to accomplish these specific training tasks. Identify visual cues used to support these methods that are found within real world terrains that do not exist within our simulator. At this point, you now have a wish list of unmet terrain visualization perceptual stimuli requirements that you aspire to implement within GENETICS. After implementation, perform tests to see if the visual cues added with the GENETICS system actually aid in the training of the desired task. Success or failure with the new system further defines the visual fidelity levels needed to satisfy the perceptual requirements of the training tasks and helps to flesh out the training requirement/visual fidelity matrix described above.

APPENDIX A. COMMERCIAL IMPLEMENTATIONS

Introduction

None of the following commercial terrain vegetation implementations are currently used within the MOVES Institute. Descriptions and comments about each were gathered from vendor materials, product reviews, and the U.S. Army's Topographical Engineering Center's (TEC) Commercial Terrain Visualization Software Product Information website: <http://www.tec.army.mil/TD/tvd/survey/index.html>.

3D Nature

<http://www.3dnature.com>

3D Nature's basic product is called World Construction Set (\$500) and their premium product (includes all WCS features and adds GIS support) is called Visual Nature Studio (\$2500). Both are primarily designed as packages to support landscape creation/editing/animation and are remarkable for its ability to accept both standard GIS (e.g. DEM) and animation package file formats (e.g. 3DS, LWO) to construct a scene. Supports randomization of both "natural" and "man-made" objects (placement, size, shape, rotation) to add variety to a scene. Ecosystem support also includes both natural and man-made features & textures (e.g. "Manhattan"). Includes a real-time preview renderer with joystick support to fly-through landscapes. Supports scene exporting to animation packages like 3D Studio Max and Lightwave.

TEC description: Visual Nature Studio (VNS) is a landscape visualization product. The software extends the features of World Construction Set with GIS-oriented capabilities; e.g., it software provides tools to control visualization directly from users' GIS data. Users can import, grid and render terrain in a variety of formats, projections and datums. Remotely sensed imagery can be used to color the landscape or to control land cover themes. Users can drive or fly across virtual terrain.

VNS can convert input contour lines, survey points and breaklines into grids. A DEM Editor allows users to edit DEM values numerically. VNS allows the interpolation of higher-density elevation data where imported data aren't adequate for the application, using a Delaunay triangulation or spline operators.

Linkage to GIS data means that different vegetation or land use polygons will be rendered according to their attributes, different road or stream types will be rendered for different vectors, etc. For example, a linear feature attributed as a two-lane road will be rendered as a road with two visually distinct lanes. 3-D objects can be oriented according to the orientation of input vectors; e.g., houses along curved roads can be automatically oriented along the roads, and fence posts can automatically tilt as they go uphill. VNS can also extrude vector entities, and can render random variations of 3-D objects. Imported vector paths can be used as camera or 3-D object paths, so

users can create animations to show, for example, what an area would look like from a proposed road.

VNS includes foliage objects, textures, pre-built components and projects. Users can drag and drop skies, ecosystems, bodies of water, 3-D objects and other components into their scenes. VNS supports bump mapping to simulate spatial detail without requiring high polygon density. There are also city-like textures to simulate urban areas.

Users can link vectors to components at rendering time based on search queries. Thus, database attributes from Shapefiles can control where components are rendered. Users can select items in a database according to name, label, layer, etc. Linking vectors to components can be static or dynamic. If dynamic, VNS images can reflect the current status of the data.

VNS can import and combine data in a variety of projections, datums and coordinate systems, dynamically reprojecting data in diverse coordinate systems. VNS supports ARC/INFO ASCII DEMs, ADF, E00, ASCII arrays, DTED, GTOPO30, BIL, SDTS, MICRODEM Binary Array and United Kingdom Ordnance Survey NTF DNM formats. The software also supports DXF, JPEG, PNG, TIFF, BMP, ECW, GeoTIFF, AVI and QuickTime formats. A wizard lets users crop rows and columns dynamically as data are imported. The software supports the rendering of stereo pairs. VNS can export ArcView Shapefiles in 2-D and 3-D with attributes. There is support for external programs such as LightWave 3D, Inspire 3D and 3D Studio Max. VNS can also render trees created in Onyx Tree Professional, Tree Druid, LSYSTEMS and other programs.

VNS 2 enables users to drape images, merge DEMs of different resolutions, and apply transparent water and volumetric atmospheres. Users can expedite rendering with a network rendering controller. A new Scenario Manager has tools for managing multiple variations or alternative visualizations of the same project. Splined Terrafactors create smoothly curving river beds and roads. Terrafactor Freeze can be invoked to make terrain alterations permanent.

Blueberry 3D

<http://www.blueberry3d.com>

Blueberry3D (\$2500 + Vega licensing) is primarily geared at real-time military simulations created with commercial scene graphs like Multigen-Paradigm's Vega Prime. Recently purchased by Bionatics. It most closely represents the goal of GENETICS.

TEC description: Blueberry3D is a 3-D terrain modeling and visualization tool that consists of two applications - Blueberry3D Viewer and Blueberry3D Editor. With the Viewer, users can navigate and view virtual terrain from any direction and distance. The Editor is for creating 3-D landscapes. Users can import real-world data, or make artificial terrain.

Instead of using a limited set of discrete levels of detail (LODs), Blueberry3D uses mathematical fractals to model the ground surface and vegetation. Details are not stored explicitly. As a user zooms in, fractals automatically generate additional detail. This technique allows unlimited LODs. By computing details such as trees at run

time, memory usage is minimized. A "deep-freeze" (e.g. non-real-time) function supports photorealistic rendering of desired scenes from any perspective. This function iterates the fractals further to create finer details, and to compute soft shadows.

Fractals adjust the ground level between sample points in a height raster, and smoothen the transition between terrain classes (e.g., forests, meadows and lakes). The software uses input textures (e.g., orthophotos) and also generates its own textures for LOD management. Linear features (roads, rivers, fences, etc.) are automatically converted to smooth curves during rendering, and the surrounding terrain is appropriately leveled. Users can import 3-D models and place them in the landscape at specified coordinates. Blueberry3D includes several pre-defined trees and bushes, and users can also design their own. Vegetation is described by parameters such as average height, trunk diameter, leaf density, etc. The software also supports time- of-day effects and atmospheric effects (wind, waves, reflections, haze, fog), shadows, etc.

An Entity Subsystem lets users add dynamic content to virtual worlds. A Physics Engine lets users design and simulate tanks, helicopters, etc. Objects can interact in a natural fashion with the terrain and with other objects. Users can also add customized components, such as explosions. There are also entities to control wind, rain and other weather effects. Wind effects can include random gusts that affect vegetation and even blow leaves off trees. An optional API lets users design advanced effects and add HLA actors.

The Editor works with height rasters, terrain class rasters, textures, vector data and 3-D models. The software supports ARC/INFO BIL, ASCII GRID and binary GRID; ArcView Shape; USGS DOQ and SDTS DEM; NGA DTED; Portable Gray Map (PGM); Windows Bitmap (BMP); Tagged Image File (TIFF); GeoTIFF; JPEG; GIF; PNG and TARGA input formats. The database is stored implicitly as mathematical formulas that are used to compute geometry at run time. The software exports Windows 24-bit RGB bitmaps (BMP). Users can also define camera paths through 3-D worlds to create movies.

Blueberry3D was demonstrated at [TEC](#) in October 2001. In one scenario a tank was driven through a natural-looking virtual world that included grass, hills, roads, fences, buildings, and trees with swaying limbs and blowing leaves. Capabilities to perform line-of-sight calculations, radio link calculations, path finding (shortest, fastest, cheapest) and infrared (IR) and night vision goggle (NVG) rendering were claimed but not demonstrated.

Descendor

<http://www.binaryworlds.com/products.html>

Descendor advertises itself as a real-time procedural world generation engine. Since 3D models are created automatically by the engine, Descendor worlds can be as big and detailed as required. For instance, you can have a galaxy model, "descend" down to a planet, approach a continent, a region, a city, enter a building and watch a flower in a room. Worlds are created randomly, but the parameters' constraints and

the creation rules ensure that they are still realistic. Descensor generates the model on-demand and only visible world portions are calculated for each frame. Objects are created at the appropriate level of detail, closer objects are rendered with greater detail.

The engine adapts to the available CPU and memory resources. The detail level is adjusted to ensure a constant frame rate while not exceeding the allotted memory limit.

Descensor is designed to generate landscapes for computer games. It is a game middleware component that integrates with other game software. It has an object-oriented design and is written in portable platform-independent ANSI C++. The world creation engine core is independent of the rendering, therefore it can use different graphics libraries like DirectX or OpenGL.

“Simulators, shooters, role-play and other kinds of games can benefit from Descensor. The engine is especially suited for online games because it allows to re-create the same world on each player's machine with minimal bandwidth usage.” Descensor model are dynamically generated and the game can modify the world. That means that if a game requires it, a player could alter the landscape, create new cities, modify buildings, etc.

GScape

<http://www.gscape.com>

No longer available, GScape was a terrain editor and visualization system (exposed by API) for an unnamed fantasy MMORPG. The system used two images, a heightmap and ecotope/vegetation map, to generate a realistic 3D landscape. Procedurally-based detailing created elevation and texture details with respect to the player's position. No explanation of the terrain LOD scheme was given. Vegetation was placed probabilistically within the terrain (as determined by the vegetation map) and used a combination of geometry, imposters, and billboards to represent the vegetation objects. Shadows were generated for most plant objects and self-shadowing of the terrain occurred. GScape offered an API of their v1.0 system, but the system was not open source and thus no source code, other than programming examples, was ever provided.

OnyxTREE PRO

<http://www.onyxtree.com>

OnyxTREE has a reputation as being an easy-to-use tree generation package. However, the suite is broken into multiple programs: one for broadleaf trees, one for conifers, one for palms, and one for bamboo. Each standalone generator can output trees in a wide array of standard output formats. While each program contains a library of exist tree species, it is easy to modify these trees (or create wholly new ones) using logically named parameters and instant on-screen feedback. Unlike SpeedTree, leaves face in every direction and are built from photorealistic textures, allowing for close inspection. Trees can be deformed by wind. Trees can also be pruned or have details eliminated, which also helps in reducing polygon count (for

example, ignoring leaf stems reduced a tree's polygon count from 151,000 to 71,000). In addition to exporting tree geometry, tree images can be exported for use in billboarding. This is likely the only way to support real-time simulation use since OnyxTREE is primarily geared towards the animation market.

SpeedTree

<http://www.idvinc.com>

SpeedTree is an interesting hybrid that is part 3D tree program, part 2D map utility, part animation package plug-in and part real-time simulation environment generator. Trees within the system (when using geometry vs billboards or imposters) range from 2,000 to 200,000 polygons. Obviously, the higher polygon count trees are used for animation purposes while the lower polygon count models are suitable for high resolution tree models within real-time sims. Clusters of leaves are applied to trees as billboards to further lower the polygon count. Leaf clusters cannot be randomized, only scaled. Thus, this means that individual leaves are not as detailed and patterns emerge, resulting in artificialities to the trees when looked at closely. Also, like most billboards, the leaves stay perpendicular to the field of view which is quite unnatural. SpeedTree does support wind-based animation of trees. The system also includes a tree designer, a shadowmap maker, and a forest creator (managing multiple level of details).

Terragen

<http://www.planetside.co.uk>

Terragen (free for noncommercial use) is a non-realtime landscape rendering system. Through procedural modeling and raytracing techniques, Terragen is able to produce stunningly realistic images of landscapes with plausible terrain ecotopes depicting believable surface coloration, detailing, shadows, clouds, atmospheric effects, lighting, and water & caustic effects. Terragen has no capability to include objects within the terrain to include vegetation. Typically, such objects are either Photoshopped into the scene or the Terragen scene is exported to an animation/rendering package like 3DS Max.

TEC description: Terragen is a scenery generator. Users can create landscapes with mountains, valleys, water, sunlight, clouds, shadows, haze, etc. Terrain sculpting tools are included, and random terrain generation can be invoked. Terragen supports landscapes up to 4097 x 4097, with real-world scaling (allowing positions to be specified in meters.) Users can choose any viewpoint in a 3-D scene, and can find the altitude of any point on the terrain.

Users can control parameters to generate fractal terrain, which can be used to add realistic details. Terragen's rendering engine has automatic level-of-detail adjustment. There are terrain sculpting tools, and terrain modification tools to create effects such as glaciation. Two terrains can be combined, using several different methods. The surface of a landscape can be divided into different components (e.g., grass and rock) to create a hierarchical surface color map. These components can be subdivided

further, as desired. Terragen can render bodies of water with ripples, waves and soft reflections. There is a cloud generator, and a sunlight penetration system that calculates the dimming and reddening of sunlight through the atmosphere.

Terragen can import and export raw height field information in 8-bit grayscale. When a user opens a file, a grayscale image appears in the Landscape and Image windows. Hitting the Render Preview button converts the grayscale image into a 3-D elevation model.

There are utilities available for converting USGS DEM files for import into Terragen. For example, the program SDTSTER4 (available from www.terrainmap.com) converts SDTS DEM files to the TER format required by Terragen. The program MDEM2TER (also available from www.terrainmap.com) converts from MicroDEM native DEM format to the TER format. With this utility, Terragen can ingest any of the numerous DEM formats that MicroDem can. In the TER format, elevation values are represented as a series of two- byte integers.)

Terragen terrain files can be exported as BMP files, LightWave 3D Object (LWO) files, 8-bit binary files, and binary files compatible with VistaPro software. Plug-ins are available to facilitate integration of Terragen scenery into LightWave scenes and animations.

TreeMagik Pro

<http://www.aliencodec.com>

Treemagik Pro is a low polygon count tree generator that is ideally suited for game development. Photorealistic textures are used for bark, limb and leaf clusters and trees can be exported to a large number of commonly-used game object file formats. TreeeMagik Pro is extremely inexpensive (\$45) compared to it's rivals, but lacks some of the features (animation and high polygon count) needed for detailed renderings and animation projects.

Virtual Trees and Foliage

<http://www.marlinstudios.com>

Marlin Studios has produced an package of over 200 tree and foliage textures that can be applied to billboards for use within animation packages and real-time simulations. Unfortunately, like most billboard textures, the trees all have preset lighting, generally with the sun high in the sky. This is needed to bring out the contrast in the leaves, but may look awkward depending on the application's lighting scheme. Shadow-casting versions of the tree images are also included with the package. Each tree and plant comes in three resolutions and includes bump maps and an alpha map (used to create transparency).

Vue

<http://www.e-onsoftware.com>

TEC description: Vue4 (\$200) is a tool for creating and animating 3-D scenery. Users can model 3-D terrain and render vegetation, rocks, stars, the atmosphere, etc.

The software also supports animation, motion blur, soft shadows, blurred reflections and transparencies. Vue4 can import static files from Poser software and can exchange data with 3D Studio MAX and LightWave software.

Xfrog

<http://www.greenworks.de>

Greenworks scientists and engineers have published several academic papers on the generation of a wide variety of plants. With Xfrog (\$330), this academic work has been productized into an organic form construction kit. Xfrog allows the user to choose the number and type of features for the plant which gives the plant designer a large range of tuneable parameters to construct and modify the plant's appearance. Unfortunately, the system suffers from a less-than-intuitive interface. The plants created with Xfrog (or supplied with their libraries) are impressively detailed and can respond to deformer input that can simulate wind or the growth of the plant. Like OnyxTREE, this system is primarily intended for exporting high polygon count vegetation objects to an animation package.

Other plants

<http://www.vterrain.org/Plants/plantsw.html>

In addition to TEC's website on terrain visualization, the Virtual Terrain Project website contains several other references to both commercial and non-commercial implementations of terrain & vegetation visualization programs.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. CONSISTENCY/VARIATION TEST DATA

Note below the consistency of GENETICS output on two different machines with and without cached data. The end of output describes the position (“P”), scaling factors (“S”), and orientation (“O”) of all the objects within the last leaf group of the scene graph. The following pages show additional test results using different parameters.

Test: Gargovle, seed 42, 10 looks, low-res, non-cached

```
dtCore-Info:MaxTextureSize = 1024
dtCore-Info:Random seed = 42
dtCore-Info:Looks per pixel = 10
dtCore-Info:SOARXTerrain: Making detail gradient image for level 1...
dtCore-Info:SOARXTerrain: Making detail scale image for level 1...
dtCore-Info:SOARXTerrain: Making base gradient image for w122_n36_1...
dtCore-Info:SOARXTerrain: Making base color image for w122_n36_1...
dtCore-Info:SOARXTerrain: Making base LCC color image for w122_n36_1...
dtCore-Info:SOARXTerrain: Making LCC image for LCC type 41...
dtCore-Info:SOARXTerrain: Making LCC smoothed image for LCC type 41...
dtCore-Info:SOARXTerrain: Making LCC image for LCC type 42...
dtCore-Info:SOARXTerrain: Making LCC smoothed image for LCC type 42...
dtCore-Info:SOARXTerrain: Making LCC image for LCC type 43...
dtCore-Info:SOARXTerrain: Making LCC smoothed image for LCC type 43...
dtCore-Info:SOARXTerrain: Making LCC image for LCC type 51...
dtCore-Info:SOARXTerrain: Making LCC smoothed image for LCC type 51...
dtCore-Info:SOARXTerrain: Making heightmap image for level 1...
dtCore-Info:SOARXTerrain: Making slopemap image for level 1...
dtCore-Info:SOARXTerrain: Making relative elevation image for level 1...
dtCore-Info:SOARXTerrain: Making probability map for LCC type 41.
dtCore-Info:SOARXTerrain: Making probability map for LCC type 42.
dtCore-Info:SOARXTerrain: Making probability map for LCC type 43.
dtCore-Info:SOARXTerrain: Making probability map for LCC type 51.
dtCore-Info:SOARXTerrain: Making base map for w122_n36_1...
dtCore-Info:SOARXTerrain: Making detail map...
dtCore-Info:SOARXTerrain: Loaded w122_n36_1
dtCore-Info:Placing LCtype 51 'shrubland'...
dtCore-Info:shrubland count = 377574
dtCore-Info:Placing LCtype 43 'mixed forest'...
dtCore-Info:mixed forest count = 144173
dtCore-Info:Placing LCtype 42 'evergreen'...
dtCore-Info:evergreen count = 530427
dtCore-Info:Placing LCtype 41 'deciduous'...
dtCore-Info:deciduous count = 23771
dtCore-Info:Total count = 1075945
dtCore-Info:P#0, 41965.011719, 46248.007813, 169.135163
dtCore-Info:P#1, 41993.089844, 46238.347656, 174.253027
dtCore-Info:P#2, 41908.214844, 46247.980469, 165.889690
dtCore-Info:P#3, 41968.640625, 46258.378906, 171.022928
dtCore-Info:P#4, 41911.152344, 46274.613281, 167.778138
dtCore-Info:P#5, 41657.773438, 46329.535156, 154.939283
dtCore-Info:P#6, 41595.351563, 46328.160156, 147.958816
dtCore-Info:P#7, 41623.378906, 46384.621094, 154.800379
dtCore-Info:P#8, 41908.539063, 46432.367188, 172.624299
dtCore-Info:S#0, 2.272305, 2.272305, 1.672305
dtCore-Info:S#1, 1.888135, 1.888135, 1.288135
dtCore-Info:S#2, 1.864011, 1.864011, 1.264011
dtCore-Info:S#3, 1.959592, 1.959592, 1.359592
dtCore-Info:S#4, 1.920148, 1.920148, 1.320148
dtCore-Info:S#5, 1.830197, 1.830197, 1.230197
dtCore-Info:S#6, 2.078732, 2.078732, 1.478732
dtCore-Info:S#7, 1.893735, 1.893735, 1.293735
dtCore-Info:S#8, 1.847333, 1.847333, 1.247333
dtCore-Info:Q#0, 6.257577
dtCore-Info:Q#1, 2.038394
dtCore-Info:Q#2, 0.448845
dtCore-Info:Q#3, 6.233046
dtCore-Info:Q#4, 2.650525
dtCore-Info:Q#5, 5.948062
dtCore-Info:Q#6, 2.588239
dtCore-Info:Q#7, 0.690133
dtCore-Info:Q#8, 3.644424
```

Test: Genetics, seed 42, 10 looks, low-res, cached

```
dtCore-Info:MaxTextureSize = 1024
dtCore-Info:Random seed = 42
dtCore-Info:Looks per pixel = 10
dtCore-Info:SOARXTerrain: Loaded w122_n36_1
dtCore-Info:Placing LCtype 51 'shrubland'...
dtCore-Info:shrubland count = 377574
dtCore-Info:Placing LCtype 43 'mixed forest'...
dtCore-Info:mixed forest count = 144173
dtCore-Info:Placing LCtype 42 'evergreen'...
dtCore-Info:evergreen count = 530427
dtCore-Info:Placing LCtype 41 'deciduous'...
dtCore-Info:deciduous count = 23771
dtCore-Info:Total count = 1075945
dtCore-Info:P#0, 41965.011719, 46248.007813, 169.135163
dtCore-Info:P#1, 41993.089844, 46238.347656, 174.253027
dtCore-Info:P#2, 41908.214844, 46247.980469, 165.889690
dtCore-Info:P#3, 41968.640625, 46258.378906, 171.022928
dtCore-Info:P#4, 41911.152344, 46274.613281, 167.778138
dtCore-Info:P#5, 41657.773438, 46329.535156, 154.939283
dtCore-Info:P#6, 41595.351563, 46328.160156, 147.958816
dtCore-Info:P#7, 41623.378906, 46384.621094, 154.800379
dtCore-Info:P#8, 41908.539063, 46432.367188, 172.624299
dtCore-Info:S#0, 2.272305, 2.272305, 1.672305
dtCore-Info:S#1, 1.888135, 1.888135, 1.288135
dtCore-Info:S#2, 1.864011, 1.864011, 1.264011
dtCore-Info:S#3, 1.959592, 1.959592, 1.359592
dtCore-Info:S#4, 1.920148, 1.920148, 1.320148
dtCore-Info:S#5, 1.830197, 1.830197, 1.230197
dtCore-Info:S#6, 2.078732, 2.078732, 1.478732
dtCore-Info:S#7, 1.893735, 1.893735, 1.293735
dtCore-Info:S#8, 1.847333, 1.847333, 1.247333
dtCore-Info:Q#0, 6.257577
dtCore-Info:Q#1, 2.038394
dtCore-Info:Q#2, 0.448845
dtCore-Info:Q#3, 6.233046
dtCore-Info:Q#4, 2.650525
dtCore-Info:Q#5, 5.948062
dtCore-Info:Q#6, 2.588239
dtCore-Info:Q#7, 0.690133
dtCore-Info:Q#8, 3.644424
```

Test: Genetics, seed 47, 10 looks, low-res, non-cached

dtCore-Info:MaxTextureSize = 1024
dtCore-Info:Random seed = 47
dtCore-Info:Looks per pixel = 10
dtCore-Info:SOARXTerrain: Making detail gradient image for level 1...
dtCore-Info:SOARXTerrain: Making detail scale image for level 1...
dtCore-Info:SOARXTerrain: Making base gradient image for w122_n36_1...
dtCore-Info:SOARXTerrain: Making base color image for w122_n36_1...
dtCore-Info:SOARXTerrain: Making base LCC color image for w122_n36_1...
dtCore-Info:SOARXTerrain: Making LCC image for LCC type 41...
dtCore-Info:SOARXTerrain: Making LCC smoothed image for LCC type 41...
dtCore-Info:SOARXTerrain: Making LCC image for LCC type 42...
dtCore-Info:SOARXTerrain: Making LCC smoothed image for LCC type 42...
dtCore-Info:SOARXTerrain: Making LCC image for LCC type 43...
dtCore-Info:SOARXTerrain: Making LCC smoothed image for LCC type 43...
dtCore-Info:SOARXTerrain: Making LCC image for LCC type 51...
dtCore-Info:SOARXTerrain: Making LCC smoothed image for LCC type 51...
dtCore-Info:SOARXTerrain: Making heightmap image for level 1...
dtCore-Info:SOARXTerrain: Making slopemap image for level 1...
dtCore-Info:SOARXTerrain: Making relative elevation image for level 1...
dtCore-Info:SOARXTerrain: Making probability map for LCC type 41.
dtCore-Info:SOARXTerrain: Making probability map for LCC type 42.
dtCore-Info:SOARXTerrain: Making probability map for LCC type 43.
dtCore-Info:SOARXTerrain: Making probability map for LCC type 51.
dtCore-Info:SOARXTerrain: Making base map for w122_n36_1...
dtCore-Info:SOARXTerrain: Making detail map...
dtCore-Info:SOARXTerrain: Loaded w122_n36_1
dtCore-Info:Placing LCCtype 51 'shrubland'...
dtCore-Info:shrubland count = 377212
dtCore-Info:Placing LCCtype 43 'mixed forest'...
dtCore-Info:mixed forest count = 144296
dtCore-Info:Placing LCCtype 42 'evergreen'...
dtCore-Info:evergreen count = 530786
dtCore-Info:Placing LCCtype 41 'deciduous'...
dtCore-Info:deciduous count = 23862
dtCore-Info:Total count = 1076156
dtCore-Info:P#0, 41717.445313, 46296.128906, 156.095442
dtCore-Info:P#1, 41940.195313, 46268.972656, 169.734971
dtCore-Info:P#2, 41900.257813, 46242.878906, 166.291203
dtCore-Info:P#3, 41970.937500, 46256.710938, 171.283750
dtCore-Info:P#4, 41658.574219, 46344.796875, 157.149642
dtCore-Info:P#5, 41653.929688, 46412.203125, 161.656908
dtCore-Info:P#6, 41589.968750, 46339.500000, 147.590618
dtCore-Info:P#7, 41681.917969, 46326.003906, 156.817866
dtCore-Info:P#8, 41923.832031, 46420.507813, 175.444012
dtCore-Info:S#0, 1.884900, 1.884900, 1.284900
dtCore-Info:S#1, 2.062894, 2.062894, 1.462894
dtCore-Info:S#2, 2.150372, 2.150372, 1.550372
dtCore-Info:S#3, 1.817151, 1.817151, 1.217151
dtCore-Info:S#4, 1.925778, 1.925778, 1.325778
dtCore-Info:S#5, 1.810346, 1.810346, 1.210346
dtCore-Info:S#6, 1.807172, 1.807172, 1.207172
dtCore-Info:S#7, 1.819180, 1.819180, 1.219180
dtCore-Info:S#8, 2.066098, 2.066098, 1.466098
dtCore-Info:Q#0, 3.522343
dtCore-Info:Q#1, 2.019037
dtCore-Info:Q#2, 2.780656
dtCore-Info:Q#3, 4.921007
dtCore-Info:Q#4, 3.931133
dtCore-Info:Q#5, 2.410770
dtCore-Info:Q#6, 3.624109
dtCore-Info:Q#7, 5.637588
dtCore-Info:Q#8, 5.155012

Test: Voodoo, seed 47, 10 looks, low-res, cached

dtCore-Info:MaxTextureSize = 1024
dtCore-Info:Random seed = 47
dtCore-Info:Looks per pixel = 10
dtCore-Info:SOARXTerrain: Loaded w122_n36_1
dtCore-Info:Placing LCCtype 51 'shrubland'...
dtCore-Info:shrubland count = 377212
dtCore-Info:Placing LCCtype 43 'mixed forest'...
dtCore-Info:mixed forest count = 144296
dtCore-Info:Placing LCCtype 42 'evergreen'...
dtCore-Info:evergreen count = 530786
dtCore-Info:Placing LCCtype 41 'deciduous'...
dtCore-Info:deciduous count = 23862
dtCore-Info:Total count = 1076156
dtCore-Info:P#0, 41717.445313, 46296.128906, 156.095442
dtCore-Info:P#1, 41940.195313, 46268.972656, 169.734971
dtCore-Info:P#2, 41900.257813, 46242.878906, 166.291203
dtCore-Info:P#3, 41970.937500, 46256.710938, 171.283750
dtCore-Info:P#4, 41658.574219, 46344.796875, 157.149642
dtCore-Info:P#5, 41653.929688, 46412.203125, 161.656908
dtCore-Info:P#6, 41589.968750, 46339.500000, 147.590618
dtCore-Info:P#7, 41681.917969, 46326.003906, 156.817866
dtCore-Info:P#8, 41923.832031, 46420.507813, 175.444012
dtCore-Info:S#0, 1.884900, 1.884900, 1.284900
dtCore-Info:S#1, 2.062894, 2.062894, 1.462894
dtCore-Info:S#2, 2.150372, 2.150372, 1.550372
dtCore-Info:S#3, 1.817151, 1.817151, 1.217151
dtCore-Info:S#4, 1.925778, 1.925778, 1.325778
dtCore-Info:S#5, 1.810346, 1.810346, 1.210346
dtCore-Info:S#6, 1.807172, 1.807172, 1.207172
dtCore-Info:S#7, 1.819180, 1.819180, 1.219180
dtCore-Info:S#8, 2.066098, 2.066098, 1.466098
dtCore-Info:Q#0, 3.522343
dtCore-Info:Q#1, 2.019037
dtCore-Info:Q#2, 2.780656
dtCore-Info:Q#3, 4.921007
dtCore-Info:Q#4, 3.931133
dtCore-Info:Q#5, 2.410770
dtCore-Info:Q#6, 3.624109
dtCore-Info:Q#7, 5.637588
dtCore-Info:Q#8, 5.155012

Test: Voodoo, seed 12, 10 looks, low-res, non-cached

dtCore-Info:MaxTextureSize = 1024
dtCore-Info:Random seed = 12
dtCore-Info:Looks per pixel = 10
dtCore-Info:SOARXTerrain: Making detail gradient image for level 1...
dtCore-Info:SOARXTerrain: Making detail scale image for level 1...
dtCore-Info:SOARXTerrain: Making base gradient image for w122_n36_1...
dtCore-Info:SOARXTerrain: Making base color image for w122_n36_1...
dtCore-Info:SOARXTerrain: Making base LCC color image for w122_n36_1...
dtCore-Info:SOARXTerrain: Making LCC image for LCC type 41...
dtCore-Info:SOARXTerrain: Making LCC smoothed image for LCC type 41...
dtCore-Info:SOARXTerrain: Making LCC image for LCC type 42...
dtCore-Info:SOARXTerrain: Making LCC smoothed image for LCC type 42...
dtCore-Info:SOARXTerrain: Making LCC image for LCC type 43...
dtCore-Info:SOARXTerrain: Making LCC smoothed image for LCC type 43...
dtCore-Info:SOARXTerrain: Making LCC image for LCC type 51...
dtCore-Info:SOARXTerrain: Making LCC smoothed image for LCC type 51...
dtCore-Info:SOARXTerrain: Making heightmap image for level 1...
dtCore-Info:SOARXTerrain: Making slopemap image for level 1...
dtCore-Info:SOARXTerrain: Making relative elevation image for level 1...
dtCore-Info:SOARXTerrain: Making probability map for LCC type 41...
dtCore-Info:SOARXTerrain: Making probability map for LCC type 42...
dtCore-Info:SOARXTerrain: Making probability map for LCC type 43...
dtCore-Info:SOARXTerrain: Making probability map for LCC type 51...
dtCore-Info:SOARXTerrain: Making base map for w122_n36_1...
dtCore-Info:SOARXTerrain: Making detail map...
dtCore-Info:SOARXTerrain: Loaded w122_n36_1
dtCore-Info:Placing LCtype 51 'shrubland'...
dtCore-Info:shrubland count = 376854
dtCore-Info:Placing LCtype 43 'mixed forest'...
dtCore-Info:mixed forest count = 144100
dtCore-Info:Placing LCtype 42 'evergreen'...
dtCore-Info:evergreen count = 530477
dtCore-Info:Placing LCtype 41 'deciduous'...
dtCore-Info:deciduous count = 23862
dtCore-Info:Total count = 1075293
dtCore-Info:P#0, 41983.542969, 46239.863281, 172.374678
dtCore-Info:P#1, 41970.988281, 46271.304688, 172.685638
dtCore-Info:P#2, 41969.351563, 46266.960938, 171.857613
dtCore-Info:P#3, 41918.757813, 46263.863281, 167.342819
dtCore-Info:P#4, 41640.316406, 46315.039063, 151.228281
dtCore-Info:P#5, 41613.078125, 46400.718750, 152.880629
dtCore-Info:P#6, 41632.515625, 46355.152344, 153.080794
dtCore-Info:P#7, 41656.335938, 46377.492188, 160.645927
dtCore-Info:P#8, 41655.769531, 46401.285156, 161.575840
dtCore-Info:P#9, 41736.910156, 46479.640625, 169.648288
dtCore-Info:S#0, 2.156918, 2.156918, 1.556918
dtCore-Info:S#1, 2.237836, 2.237836, 1.637836
dtCore-Info:S#2, 2.270596, 2.270596, 1.670596
dtCore-Info:S#3, 1.923901, 1.923901, 1.323901
dtCore-Info:S#4, 1.861951, 1.861951, 1.261951
dtCore-Info:S#5, 2.148541, 2.148541, 1.548541
dtCore-Info:S#6, 2.131635, 2.131635, 1.531635
dtCore-Info:S#7, 2.264172, 2.264172, 1.664172
dtCore-Info:S#8, 2.192548, 2.192548, 1.592548
dtCore-Info:S#9, 2.200894, 2.200894, 1.600894
dtCore-Info:Q#0, 3.128118
dtCore-Info:Q#1, 3.057782
dtCore-Info:Q#2, 6.197591
dtCore-Info:Q#3, 1.474367
dtCore-Info:Q#4, 4.755805
dtCore-Info:Q#5, 6.253552
dtCore-Info:Q#6, 1.275050
dtCore-Info:Q#7, 0.344971
dtCore-Info:Q#8, 1.091257
dtCore-Info:Q#9, 0.480084

Test: Gargovle, seed 12, 10 looks, low-res, cached

dtCore-Info:MaxTextureSize = 1024
dtCore-Info:Random seed = 12
dtCore-Info:Looks per pixel = 10
dtCore-Info:SOARXTerrain: Loaded w122_n36_1
dtCore-Info:Placing LCtype 51 'shrubland'...
dtCore-Info:shrubland count = 376854
dtCore-Info:Placing LCtype 43 'mixed forest'...
dtCore-Info:mixed forest count = 144100
dtCore-Info:Placing LCtype 42 'evergreen'...
dtCore-Info:evergreen count = 530477
dtCore-Info:Placing LCtype 41 'deciduous'...
dtCore-Info:deciduous count = 23862
dtCore-Info:Total count = 1075293
dtCore-Info:P#0, 41983.542969, 46239.863281, 172.374678
dtCore-Info:P#1, 41970.988281, 46271.304688, 172.685638
dtCore-Info:P#2, 41969.351563, 46266.960938, 171.857613
dtCore-Info:P#3, 41918.757813, 46263.863281, 167.342819
dtCore-Info:P#4, 41640.316406, 46315.039063, 151.228281
dtCore-Info:P#5, 41613.078125, 46400.718750, 152.880629
dtCore-Info:P#6, 41632.515625, 46355.152344, 153.080794
dtCore-Info:P#7, 41656.335938, 46377.492188, 160.645927
dtCore-Info:P#8, 41655.769531, 46401.285156, 161.575840
dtCore-Info:P#9, 41736.910156, 46479.640625, 169.648288
dtCore-Info:S#0, 2.156918, 2.156918, 1.556918
dtCore-Info:S#1, 2.237836, 2.237836, 1.637836
dtCore-Info:S#2, 2.270596, 2.270596, 1.670596
dtCore-Info:S#3, 1.923901, 1.923901, 1.323901
dtCore-Info:S#4, 1.861951, 1.861951, 1.261951
dtCore-Info:S#5, 2.148541, 2.148541, 1.548541
dtCore-Info:S#6, 2.131635, 2.131635, 1.531635
dtCore-Info:S#7, 2.264172, 2.264172, 1.664172
dtCore-Info:S#8, 2.192548, 2.192548, 1.592548
dtCore-Info:S#9, 2.200894, 2.200894, 1.600894
dtCore-Info:Q#0, 3.128118
dtCore-Info:Q#1, 3.057782
dtCore-Info:Q#2, 6.197591
dtCore-Info:Q#3, 1.474367
dtCore-Info:Q#4, 4.755805
dtCore-Info:Q#5, 6.253552
dtCore-Info:Q#6, 1.275050
dtCore-Info:Q#7, 0.344971
dtCore-Info:Q#8, 1.091257
dtCore-Info:Q#9, 0.480084

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. NETWORKED PLAYERS TEST

In Figure C-1, note how the simple addition of a detailed tree object can almost completely obscure a nearby player within a distributed virtual environment. Without cover to hide behind, the tank is clearly exposed to the helicopter and vice versa. This test also demonstrates the consistency of the GENETICS-generated synthetic nature environment. The positioning, size, and type of tree in both player's views are consistent with one another, allowing for a fair fight to occur.

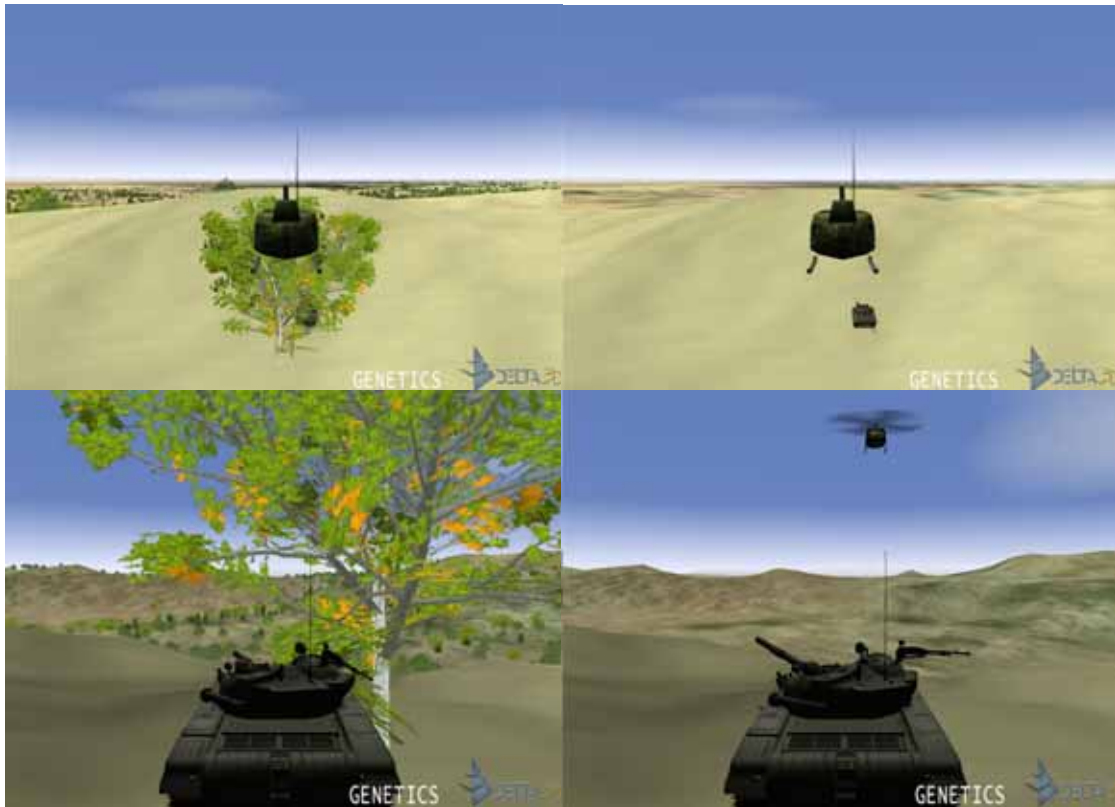


Figure C-1 Tank behind the tree test (with and without the tree)

Figure C-2 demonstrates another instance of the tank-behind-the-tree test, but this time the focus is on the relative ease of identifying a tank from a long distance away without the cover of vegetation. In the left scene, it is nearly impossible to distinguish the hidden tank from the numerous trees. In the right scene, the tank is clearly seen just above the center of the screen.

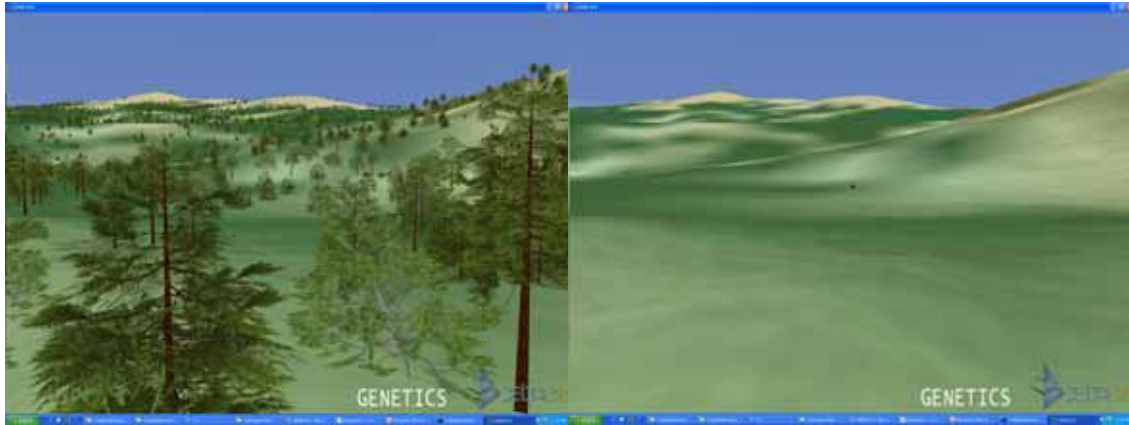


Figure C-2 Tank behind the tree test – Part 2

On the following page, Figure C-3 shows both the tank and the helicopter out in the open. The tank is much more difficult to see with the clutter surrounding it while the helicopter, in a relatively flat environment, is exposed against the clear blue sky. Two screen captures from each player's vantage point are given to compare relative distances and difficulty in detection. In the first set of pictures, both target objects are highlighted with a red circle, although the helicopter is much easier to pick out than the tank which could easily be mistaken for a bush. In the second set of pictures, the helicopter is closer and has reduced its altitude, making both tank and helicopter clearly recognizable. A further reduction in altitude would allow the helicopter to begin to blend into the vegetation behind it on the ridgeline. This situation is explored further in Figure C-4.



Figure C-3 Searching for the tank

On the following page, Figure C-4 demonstrates how helicopters can vanish into the background of a densely cluttered scene. Both pictures show the same scene from different vantage points. The tank is relatively easy to spot from the helicopter's viewpoint, but the helicopter has completely blended into its environment and while stationary, is practically indistinguishable from the background vegetation and terrain.



Figure C-4 Easy to see tank; hard to see helicopter



Figure C-5 Tank blending in with bushes

Figure C-5 demonstrates show a tank can hide in plain sight by remaining stationary within a field of bushes. This technique is show again in the series of images in Figure C-7 when the question is asked, when did you first detect the tank?



Figure C-6 Tank behind a dead bush

In Figure C-6, we again examine the consistency of the environment. Between the tank and the helicopter are a dead bush, a sweetgum, two green bushes, another sweetgum, and a sugar maple. Note the order, size, and positioning of these objects is identical.

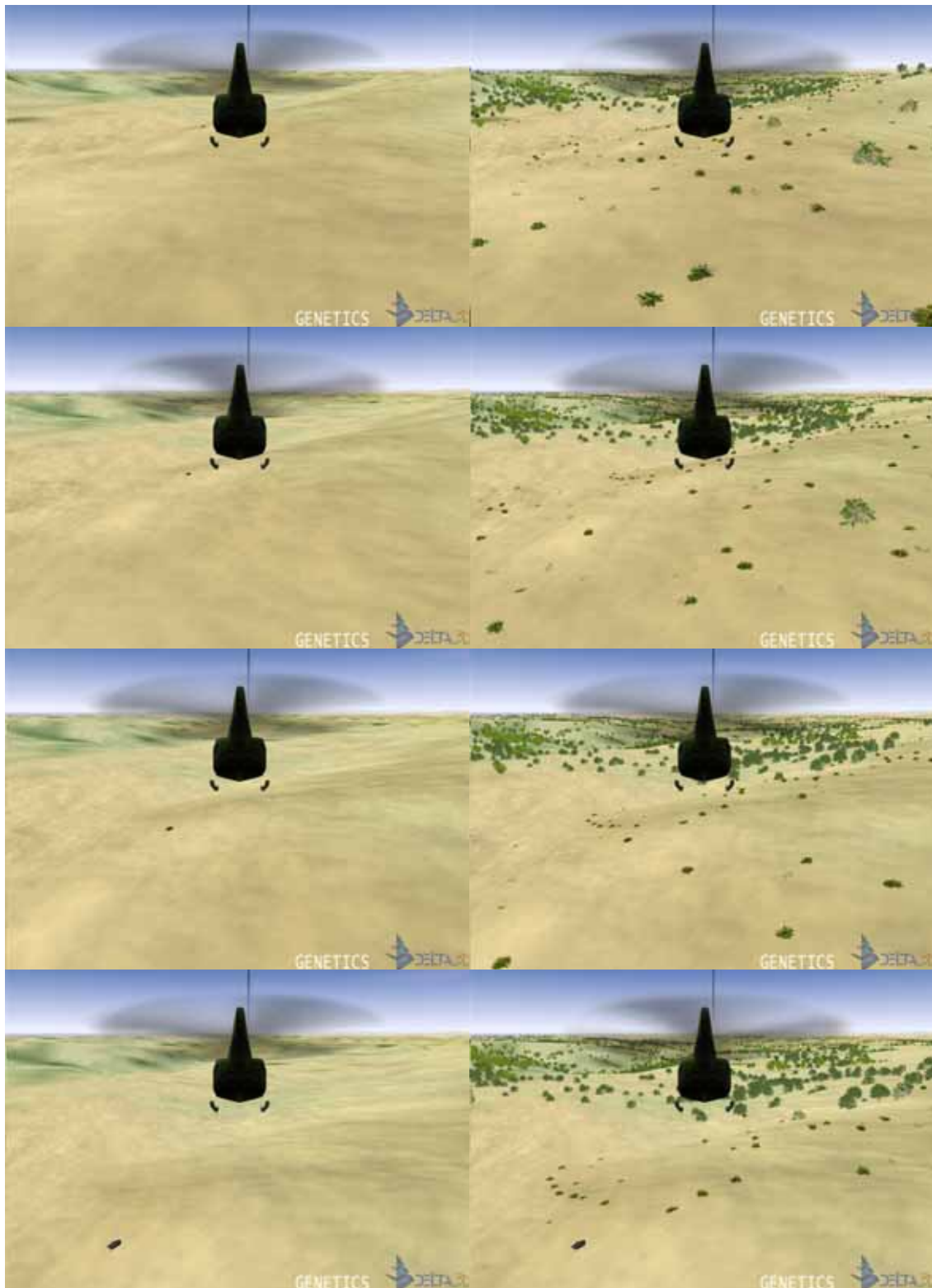


Figure C-7 Passing over the tank

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. FUTURE GENETICS RESEARCH

The following list consists of potential extensions and avenues for further research for the existing GENETICS framework:

1. Conduct a task analysis on a particular platform simulator (helicopter simulators look like a good candidate) identifying tasks, goals and methods where real-world visual cues found in the terrain typically support these training objectives. Determine missing visual cues that the GENETICS system could fill.
2. Creation of a texture splatting, Wang Tiling, or procedural texture generation algorithm to automatically place terrain textures that are smoothly blended between terrain regions (cliffs to plains, dirt to grass, etc). Texture laydown must be responsive to environment properties (cliffs look rocky, grassy plains look grassy, etc).
3. Creation of an effective method to represent vegetation objects with a minimum number of polygons. Must support scaling and simple animation (branches swaying in the wind). Promising techniques include the billboard cloud algorithm and use of point & line clouds to create trees & complex ecosystems as described in Chapter II.
4. Extend GENETICS algorithm for use within urban environments (e.g. automatic placement of furniture).
5. Creation of real-time terrain lighting effects, to include atmospheric lighting, lightmaps, and/or shadowmaps, and generation of starfield and moon (for moonlit nights). Volumetric lighting would be interesting to explore. Some method for real-time shading of trees will be needed. Starting place: techniques described in the Game Programming Gems (I & III) books.
6. Light-of-sight and sensor coverage algorithms must be researched and one must be manipulated or created that can work within the GENETICS environment. Graphics methods may help in this regard (e.g. casting an isector from the observer to the target to check on intersections and then gauge each intersected object's ability to obscure the target – ex: terrain 100%, tree 20%).
7. Generation of wind, rain, sleet, snow and other weather effects to alter the appearance of our terrain environment. This will serve as both an input into the vegetation object animation system and the appearance of texture maps (for terrain, trees, etc). Good potential for using shader programming.

8. Creation of simple 2D or 2.5D procedurally-generated clouds. Starting point: technique described in Game Programming Gems II. Clouds should create shadows on the surface of the terrain.
9. Use of vertex, object or “super” buffers (aka “über buffers”) to dramatically speed up terrain LOD algorithm by pre-storing known vertices (and noise data?) on the graphics card. The idea being that once all the vertices are stored on the graphics card, the CPU will only need to send the indices of the vertices to be rendered (thus reducing transfer costs by at least 66%).
10. Representation of dynamic changes to the terrain and how these changes will look on the local machine and then be transmitted to other clients. Will likely need to address late joiners problem through the use of a terrain server/logger.
11. Can 64-bit computing help us to dramatically speed up the above processes and improve terrain accuracy?

LIST OF REFERENCES

- Akenine-Moller, T., & Haines, E. (2002). *Real-Time Rendering* (2nd ed.). Natick, MA: A. K. Peters, Ltd.
- Baer, W., & Campbell, T. R. (2003). Rapid Terrain Database Generation using Image Differencing and 3D Terrain Editing Tools. In *Proceedings of SPIE's AeroSense: Communications and Networking Technologies and Systems*. Orlando, FL.
- Balogh, A. (2003). *Real-Time Visualization of Detailed Terrain*. Unpublished computer science masters thesis, Budapest University of Technology and Economics, Budapest, Hungary.
- Berger, D. (2003). Spectral Texturing for Real-Time Applications. In *Proceedings of International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '03)*. San Diego, CA.
- Bitters, B. (2004). Real-Time Simulation Database Generation: A Conceptual Model of the Future. In *Proceedings of the 2004 IMAGE Society Conference* (pp. 50-60). Scottsdale, AZ: The IMAGE Society.
- Bloom, C. (2000). *Terrain Texture Composition by Blending in the Frame Buffer (a.k.a. "Splatting Textures")*. Retrieved July 27, 2005, from www.cbloom.com/3d/techdocs/splatting.txt ; <http://www.gpgstudy.com/gpgiki/TextureSplatting>
- Burns, D., & Osfield, R. (2003). Open Scene Graph. In *Proceedings of the 2003 IMAGE Society Conference* (pp. 76-82). Scottsdale, AZ: The IMAGE Society.
- Carucci, F. (2005). Inside Geometry Instancing. In *GPU Gems2: Programming Techniques for High-Performance Graphics and General Purpose Computation*. New York: Addison-Wesley Professional.
- Cohen, M. F., Shade, J., Hiller, S., & Deussen, O. (2003). Wang Tiles for Image and Texture Generation. *ACM Transactions on Graphics*, 22(3), 287-294.
- Darken, R., McDowell, P., & Johnson, E. (2005). The Delta3D Open Source Game Engine. *IEEE Computer Graphics and Applications*, 25(3), 10-12.
- Davis, M. (2003). *America's Army PC Game Vision and Realization*. Monterey: MOVES Institute, Naval Postgraduate School.
- de Boer, W. H. (2000, 31 October 2000). *Fast Terrain Rendering Using Geometrical MipMapping*. Retrieved 27 July, 2005, from http://www.flipcode.com/articles/article_geomipmaps.shtml

- Decoret, X., Durand, F., Sillion, F., & Dorsey, J. (2003). Billboard Clouds for Extreme Model Simplification. *ACM Transactions on Graphics*, 22(3), 689--696.
- Deussen, O., Colditz, C., Stamminger, M., & Drettakis, G. (2002). Interactive Visualization of Complex Plant Ecosystems. In *Proceedings of the Conference on Visualization '02* (pp. 219-226): IEEE Computer Society.
- Deussen, O., Hanrahan, P., Lintermann, B., Mech, R., Pharr, M., & Prusinkiewicz, P. (1998). Realistic Modeling and Rendering of Plant Ecosystems. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)* (pp. 275-286): ACM Press.
- Deussen, O., & Lintermann, B. (1997). A Modelling Method and User Interface for Creating Plants. In *Proceedings of the Conference on Graphics Interface '97* (pp. 189-197): Canadian Information Processing Society.
- Duchaineau, M., Wolinsky, M., Sigeti, D. E., Miller, M. C., Aldrich, C., & Mineev-Weinstein, M. B. (1997). ROAMing Terrain: Real-time Optimally Adapting Meshes. In *Proceedings of the 8th Conference on Visualization '97* (pp. 81-88): IEEE Computer Society Press.
- Duda, R., Hart, P., & Stork, D. (2000). *Pattern Classification*: Wiley Interscience.
- Ebert, D. S. (2003). *Texturing & Modeling : A Procedural Approach* (3rd ed.). Amsterdam ; Boston: Academic Press.
- Ervin, S. M., & Hasbrouck, H. H. (2001). *Landscape Modeling : Digital Techniques for Landscape Visualization*. New York London: McGraw-Hill.
- Forsyth, T. (2001). Imposters: Adding Clutter. In M. A. DeLoura (Ed.), *Game Programming Gems 2* (pp. 488-496). Hingham, MA: Charles River Media.
- Forsyth, T. (2002). Unique Textures. In D. Treglia (Ed.), *Game Programming Gems 3* (pp. 459-466). Hingham, MA: Charles River Media.
- Frère, D., Vandekerckhove, J., Moons, T., & Van Gool, L. (1998). Automatic Modeling and 3D Reconstruction of Urban Buildings from Aerial Imagery. In *IEEE International Geoscience and Remote Sensing Symposium Proceedings*. Seattle.
- Frueh, C., & Zakhor, A. (2003). Constructing 3D City Models by Merging Ground-Based and Airborne Views. In *Proceedings of 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03)* (pp. 562).
- Gergel, S. E., & Turner, M. G. (2002). *Learning Landscape Ecology : A Practical Guide to Concepts and Techniques*. New York: Springer.

- Greuter, S., Parker, J., Stewart, N., & Leach, G. (2003). Real-time Procedural Generation of 'Pseudo Infinite' Cities. In *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia* (pp. 87-94): ACM Press.
- Haala, N., & Brenner, C. (1999). Extraction of Buildings and Trees in Urban Environments. *ISPRS Journal of Photogrammetry & Remote Sensing*, 54, 130-137.
- Hammes, J. (2001). *Modeling of Ecosystems as a Data Source for Real-Time Terrain Rendering*. Paper presented at the First International Symposium on Digital Earth Moving, London, UK.
- Hilts, S., & Mitchell, P. (1999). *The Woodlot Management Handbook*. Buffalo, NY: Firefly Books, Inc.
- Hirtz, P., Hoffmann, H., & Nüesch, D. (1999). Interactive 3D Landscape Visualization: Improved Realism Through Use of Remote Sensing Data and Geoinformation. In *Proceedings of Computer Graphics International 1999* (pp. 101-108). Canmore, Alberta, Canada.
- Hoglund, K. (2004). personal email communication with M. Zyda (Jan 8, 2004).
- Hoppe, H. (1996). Progressive Meshes. In *Computer Graphics (SIGGRAPH '96 proceedings)* (Vol. 30, pp. 99--108): ACM Press/Addison-Wesley Publishing Co.
- Hoppe, H. (1997). View-Dependent Refinement of Progressive Meshes. In *Computer Graphics and Interactive Techniques (SIGGRAPH '97 proceedings)* (pp. 189-198): ACM Press/Addison-Wesley Publishing Co.
- Hoppe, H. (1998). Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering. In *Proceedings of IEEE Visualization '98* (pp. 35-42).
- Krten, R. (1994). Generating Realistic Terrain. *Dr Dobbs Journal: Software Tools for the Professional Programmer*, 19(7), 26-28.
- Lecky-Thompson, G. W. (1999). Algorithms for an Infinite Universe. GamaSutra.
- Lecky-Thompson, G. W. (2000). Predictable Random Numbers. In M. A. DeLoura (Ed.), *Game Programming Gems* (pp. 133-140). Rockland, MA: Charles River Media.
- Lin, M. C., & Gottschalk, S. (1998). Collision Detection between Geometric Models: A Survey. In *Proceedings of IMA Conference on Mathematics of Surfaces*.
- Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L., Faust, N., & Turner, G. (1996). Real-Time Continuous Level of Detail Rendering of Height Fields. *Proceedings of SIGGRAPH'96*, 109-118.

- Lindstrom, P., & Pascucci, V. (2001). Visualization of Large Terrains Made Easy. In *Proceedings of the Conference on Visualization '01* (pp. 363-371): IEEE Computer Society.
- Lindstrom, P., & Pascucci, V. (2002). Terrain Simplification Simplified: A General Framework for View-Dependent Out-of-Core Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3), 239-254.
- Lintermann, B., & Deussen, O. (1996a). Interactive Modelling and Animation of Branching Botanical Structures. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96* (pp. 139-151): Springer-Verlag New York, Inc.
- Lintermann, B., & Deussen, O. (1996b). Interactive Modelling of Branching Structures. In *ACM SIGGRAPH 96 Visual Proceedings: The Art and Interdisciplinary Programs of SIGGRAPH '96* (pp. 148): ACM Press.
- Lintermann, B., & Deussen, O. (1997). A Modelling Method and User Interface for Creating Plants. In *Proceedings of the Conference on Graphics Interfaces '97* (pp. 189-197). Kelowna, British Columbia, Canada: Canadian Information Processing Society.
- Lintermann, B., & Deussen, O. (1999). Interactive Modeling of Plants. *IEEE Computer Graphics Applications*, 19(1), 56-65.
- Luebke, D. P. (2003). *Level of Detail for 3D Graphics* (1st ed.). Boston, MA: Morgan Kaufmann Publishers.
- Mantler, S., Tobler, R. F., & Fuhrmann, A. L. (2003). *The State of the Art in Realtime Rendering of Vegetation* (Technical Report No. 2003-027). Vienna, Austria: Virtual Reality and Visualization (VRVis) Research Center.
- Milliger, M. (2002). Procedural Texturing. In D. Treglia (Ed.), *Game Programming Gems 3* (pp. 452-458). Hingham, MA: Charles River Media.
- Musgrave, F. K. (1993). *Methods for Realistic Landscape Imaging*. Unpublished computer science doctoral thesis, Yale University.
- Okabe, A., Boots, B., & Sugihara, K. (1992). *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*: Wiley & Sons.
- O'Neil, S. (2001). *A Real-Time Procedural Universe*. Retrieved July, 2005, from http://www.gamasutra.com/features/20010302/oneil_01.htm
- Pajarola, R. B. (1998). Large Scale Terrain Visualization Using The Restricted Quadtree Triangulation. In *Proceedings of IEEE Visualization '98* (pp. 19--26).

- Parish, Y., & Müller, P. (2001). Procedural Modeling of Cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)* (pp. 301-308): ACM Press.
- Peitso, L. E. (2002). *Visual Field Requirements For Precision Nap-of-the-Earth Helicopter Flight*. Unpublished computer science masters thesis, Naval Postgraduate School, Monterey, CA.
- Perlin, K. (1985). An Image Synthesizer. *ACM SIGGRAPH Computer Graphics*, 19(3), 287-296.
- Polack, T. (2002). *Focus on 3D Terrain Programming*. Indianapolis, IN: Premier Press.
- Polack, T. (2003). Building a Flexible Terrain Engine for the Future. In *Graphics Programming Methods* (pp. 165-174). Rockland, MA: Charles River Media, Inc.
- Pratt, D. R. (1993). *A Software Architecture for the Construction and Management of Real-Time Virtual Worlds*. Unpublished computer science doctoral thesis, Naval Postgraduate School, Monterey, CA.
- Preparata, M., & Shamos, I. (1985). *Computational Geometry, An Introduction*: Springer-Verlag.
- Prusinkiewicz, P. (1999). Modeling Plants and Plant Ecosystems: Recent Results and Current Open Problems. In *Proceedings of the International Conference on Computer Graphics* (pp. 110): IEEE Computer Society.
- Prusinkiewicz, P. (2000). Simulation Modeling of Plants and Plant Ecosystems. *Communications of the ACM*, 43(7), 84-93.
- Prusinkiewicz, P., Mündermann, L., Karwowski, R., & Lane, B. (2001). The Use of Positional Information in the Modeling of Plants. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)* (pp. 289-300): ACM Press.
- Remolar, I., Chover, M., Belmonte, O., Ribelles, J., & Rebollo, C. (2002). Geometric Simplification of Foliage. In *Proceedings of Eurographics '02* (pp. 397-404): Eurographics Association.
- Remolar, I., Chover, M., Belmonte, Ó., Ribelles, J., & Rebollo, C. (2002). *Real-Time Tree Rendering* (Technical Report No. DLSI 01/03/2002): Universitat Jaume.
- Ribelles, R. C. (2003). View-Dependent Multiresolution Model for Foliage. In *Proceedings of WSCG 2003* (Vol. 26, pp. 370-378).

- Shankel, J. (2000a). Fractal Terrain Generation - Fault Formation. In M. A. DeLoura (Ed.), *Game Programming Gems* (pp. 499-502). Rockland, MA: Charles River Media.
- Shankel, J. (2000b). Fractal Terrain Generation - Midpoint Displacement. In M. A. DeLoura (Ed.), *Game Programming Gems* (pp. 503-507). Rockland, MA: Charles River Media.
- Shankel, J. (2000c). Fractal Terrain Generation - Particle Deposition. In M. A. DeLoura (Ed.), *Game Programming Gems* (pp. 508-511). Rockland, MA: Charles River Media.
- Singhal, S., & Zyda, M. (1999). *Networked Virtual Environments : Design and Implementation*. Reading, MA: Addison-Wesley.
- Snook, G. (2003). *Real-Time 3D Terrain Engines using C++ and DirectX 9* (1st ed.). Hingham, Mass.: Charles River Media, Inc.
- Spolsky, J. (2004). *Joel On Software : and on diverse and occasionally related matters that will prove of interest to software developers, designers, and managers, and to those who, whether by good fortune or ill luck, work with them in some capacity*. Berkeley, CA: Apress.
- Takase, Y., Sho, N., Sone, A., & Shimiya, K. (2001). Automatic Generation of 3D City Models and Related Applications. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXIV-5/W10.
- Ulrich, T. (2002). *Rendering Massive Terrains Using Chunked Level of Detail Control*. Retrieved July 27, 2005, from <http://www.tulrich.com/geekstuff/chunklod.html>
- van den Bergen, G. (2003). *Collision Detection in Interactive 3D Environments*: Morgan Kaufmann.
- Vandekerckhove, J., Frère, D., Moons, T., & Van Gool, L. (1998). Semi-Automatic Modelling of Urban Buildings from High Resolution Aerial Imagery. In *IEEE Proceedings of Computer Graphics International 1998*. Hanover.
- Wang, H. (1961). Proving Theorems by Pattern Recognition. *Bell Systems Technical Journal*(40), 1-42.
- Wang, H. (1965). Games, logic, and computers. *Scientific American*(November 1965), 98-106.
- Wasilewski, T., Faust, N., Grimes, M., & Ribarsky, W. (2002). *Semi-Automated Landscape Feature Extraction and Modeling* (Technical Report No. GIT-GVU-02-15). Atlanta: Georgia Tech Graphics, Visualization & Usability Center.

- Wright, E. (2005). A Methodology for Assessing the Impact of Terrain Data Quality on Military Performance. In *Proceedings of 73rd Annual Military Operations Research Society Symposium*. West Point, NY.
- Wright, G. T. (2000). *Helicopter Urban Navigation Training Using Virtual Environments*. Unpublished Masters, Naval Postgraduate School, Monterey, CA.
- Zyda, M., Mayberry, A., Wardynski, C., Shilling, R., & Davis, M. (2003). The MOVES Institute's America's Army Operations Game. In *Proceedings of 2003 Symposium of Interactive 3D Graphics* (pp. 219-220). Monterey, CA: ACM Press.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Dr. Rudy Darken
Director, MOVES Institute
Naval Postgraduate School
Monterey, CA
4. Dr. Chris Darken
Department of Computer Science
Naval Postgraduate School
Monterey, CA
5. Dr. Mathias Kölsch
Department of Computer Science
Naval Postgraduate School
Monterey, CA
6. Dr. Wolfgang Baer
Department of Information Science
Naval Postgraduate School
Monterey, CA
7. Dr. Thomas Lucas
Department of Operations Research
Naval Postgraduate School
Monterey, CA
8. COL (R) Jack Thorpe, Ph.D.
San Diego, CA